UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA

Izveštaj

Kandidat: Vedran Bajić

Broj indeksa: SV10/2023

Predmet: Objektno orijentisano programiranje 2

Tema rada: Robot u lavirintu u Knososu

Mentor rada: dr Dušan Kenjić

NOVI SAD JANUAR 2025.

Sadržaj

Uvod

Opis problema

Robot je zarobljen u lavirintu u Knososu. Njegov zadatak je da izađe iz tog lavirinta krećući se po tabli koja je na početku izgenerisana. Sa robotom je zarobljen i minotaur, koji pokušava da pojede robota. Koristeći predmete koji su postavljeni po tabli u lavirintu, robot mora da pobegne iz lavirinta, bez da ga uhvati minotaur.

Tok igre

Na početku, pri pozivanju programa, korisnik kroz argumente komandne linije zadaje dimenzije lavirinta i broj predmeta koji će biti prikazani u tabeli. Sledi primer tabele, i objasnjenje simbola.

Item: None

#############U###

#.##.#.......R#.#

#.#..#...##....##

#.##.#.##...#...#

#.........#....#

##.#.#..#..###..#

#.#....##....M.##

#.#.#....#####.##

#....#....#P..#.#

#.#...#....#..#.#

##.#.##.....P.###

#....#..P.##..#.#

#....#P...#.....#

#......#.#....#.#

#P#.#.....#...#.#

#..#.......#...#

##.#....#.#P..#.#

##.#.......#P..#

#.#..#.......#.#

#.............#

#.#....#.##.P...#

#I###############

- P – predmet
- R – robot
- M – minotaur
- I – izlaz
- U – ulaz
- . - prazno polje
- # - zid

Korisnik strelicama sa tastature upravlja robotom, krećući se levo desno gore i dole, s tim da ne može da prodje kroz zid. Minotaur posle svakog robotovog poteza igra potez u nasumičnu stranu, a ukoliko taj potez jede robota, odigraće taj. Odnosno, ukoliko se nađe pored robota, poješće ga, i korisnik je izgubio.

Kada robot dodje do izlaza, pobeđuje. Ukoliko korisnik sa tastature unese slovo “q”, igra se završava.

Predmeti

Minotaur može da uništi predmet ukoliko stane na njega. Robot stajanjem na predmet dobija specijalni efekat koji traje naredna 3 poteza. Pred početak prikazivanja table lavirinta, prikazan je tekst gde piše koji je trenutni aktivni predmet, odnosno “None” ukoliko nema predmeta. Specijalnih efekata predmeta ima 4, i oni se generišu nasumično.

- Čekić: Omogućava robotu prolaz kroz zid
- Mač: Omogućava robotu ubijanje robota
- Magla rata: Smanjuje robotu vidljivost na podmatricu 3x3

- Štit: Omugaćava korisniku da se odbrani od minotaura ukoliko ga napadne

Spisak svih klasa

- Render
- Maze_generator
- Minotaur
- Robot
- Maze
- GameEngine
- Item
    - Hammer
    - Sword
    - Fog_of_war
    - Shield

Struktura izlazne datoteke

Sve klase koje se nalaze u projektu su deo svoje statičke biblioteke. Biblioteke se koriste radi ubrzavanja kompajliranja projekta, i radi ponovne upotrebe već napisanog koda. Izlaz statičke biblioteke je .lib datoteka, na primer Maze.lib.

Za svaku biblioteku je podešen zajednički izlazni direktorijum **lib folder.** MazeGame je aplikacija, i podešeno je da linker kada krene da kompajlira program, povezuje ove datoteke iz lib direktorijuma i tako napravi jednu celinu.

Projektni direktorijum sadrži pod projekte u vidu statičkih biblioteka - za svaku klasu. takođe sadrži i MazeGame folder gde je glavni projekat, i lib folder za izlazne fajlove biblioteka.

MazeGame direktorijum nakon build-ovanja sadrži x64/Release/ folder, u kojem se nalazi MazeGame.exe datoteka, kao i maze.txt koji je izlaz programa ukoliko nije došlo do grešaka.

Analiza

Ulazno izlazni podsistemi

Kretanje robota je omogućeno klikom na strelice. Funkcija **GetAsyncKeyState** iz windows.h biblioteke prima virtuelni kod tastera, i vraća vrednost tipa short. Ako je najznačajniji bit postavljen na 1, taster je pritisnut. Tako da to proveravamo bitwise operacijom & sa 0x8000, pošto short ima 16 bita:

if (GetAsyncKeyState(VK_UP) & 0x8000){

Na kretanje minotaura ne utičemo mi, on je programiran da se sam kreće u nasumično izabranom, ili da pojede robota.

Render klasa

Ovom klasom prikazujemo tablu lavirnta korisniku. Zadužena je za 2 stvari, prikaz tabele kroz konzolu, i na čuvanje poslednje slike lavirinta u tekstualni fajl kada se igra završi.

void Render::display(char board)**

Što se tiče prikaza lavirinta, metoda prima pokazivač na 2D matricu karaktera koja predstavlja lavirint. kroz konstruktor Render klase smo prosledili dužinu i širinu, pa znamo granice tabele.

void Render::save_game(char board)**

Čuvanje tabele u fajl se vrši ispisivanjem tabele u fajl koristeći fajl streamove. Tabela se čuva u datoteci maze.txt.

Maze_generator

Ovo je klasa koja nam služi da na početku programa izgenerišemo lavirint sa datim dimenzijama. Pomoćni atributi su matrica **visited** kojom obelažavamo gde smo bili pri prolasku kroz lavirint. Niz **direction** od 4 elementa koji su **std::pair<int, int>** , što predstavlja smerove levo desno gore i dole. (Na primer gore je par (-1, 0) zato što se po redovima pomeramo za –1 a po kolonama se ne pomeramo)

void Maze_generator::generate(char board)**

Metoda prima pokazivač na 2D matricu karaktera koja predstavlja lavirint. Glavna ideja je da postavljamo blokove zida na nasumična mesta, i onda proverimo da li je moguće robotu da dođe do izlaza, odnosno poslednjeg reda matrice. Koristi algoritme kao što su dfs za pretragu kroz matricu.

bool Maze_generator::find_path (int r, int c, char board)**

Ovo je metoda koja vraća bool vrednost, true ukoliko robot sa date početne pozicije može da dodje do cilja, ili false ako ne može.

Ova metoda je zapravo dfs, poznat algoritam pretrage kroz graf, samo što je primenjen na matricu.

Minotaur klasa

Ovo je klasa koja predstavlja minotaura. On ima svoje koordinate (r, c) i glavna funkcija ove klase je:

int Minotaur::play(Maze& maze)

Funkcija prima lavirint po referenci i gleda susedna polja minotaura da bi utvrdila kuda može da se kreće. Takođe koristi pomoćni niz **direction.** Ukoliko pronađe robota pored sebe, odigraće ka njemu i pojesti ga. U suprotnom, od svih dostupnih poteza koje je sačuvala u vector strukturi, bira jedan smer nasumično. Funkcija vraća 1 ukoliko minotaur postoji (nije pojeden) i ako je odigrao potez. Ukoliko nije, vratice 0.

Robot klasa

Klasa koja upravlja kretanjem robota, i njegovim dodatnim osobinama kao što su predmeti.

int Robot::play(Maze& maze)

Funkcija play prima lavirint po referenci i na osnovu susednih polja robota, kontroliše njegovo kretanje. Ulaz sa tastature je objašnjen na početku, i kada postignemo određeni ulaz sa tastature, ukoliko robot može na to polje, potez se odigra i funkcija vraća vrednost. Ako je korisnik uneo Q kroz tastaturu, to je prekid igre, i play vraća 2. Ako je robot odigrao potez, vraća 0. U ovom delu proveravamo da li robot poseduje predmet mač ili čekić, pa primenjujemo njegov efekat.

void Robot::set_random_item()

Ovom funkcijom kreiramo novi predmet za robota. Poziva se kada robot stane na predmet, odnosno karakter P. Nasumično biramo broj između 0 i 3, i na osnovu toga kreiramo predmet (0-magla rata, 1-mač, 2-čekić, 3-štit).

Maze klasa

Ova klasa je zadužena oko lavirinta, njegovom pristupu, upravljanju i slično. U konstrukturu klase se pravi Maze generator klasa koja generiše lavirint. Atributi klase su **Render** render, i **char**** board.

void Maze::display_maze()

Poziva render klasu i njenu metodu display, uz prosleđen paramter board (tabla lavirnta)

pair<int, int> Maze::get_position(const char target)

Ova metoda prolazi kroz lavirint i vraća poziciju gde se nalazi zadati karakter. Koristi se za jedinstvene pozicije tako da nema preklapanja, postoji samo jedan takav karakter na celoj tabli. Koristimo je za minotaura, robota, izlaz... povratna vrednost je pair, pozicija r, c.

char*& Maze::operator[](int row)

Preklopljen operator pristupa lavirintu. Vraća poziciju u 2d matrici board.

Game_engine

Ova klasa upravlja celom strukturom igrice. Njena metoda run se poziva iz maina, i kontrolise korisnika, lavirint, minotaura, prekid i kraj igre...

Njeni atributi su tipa: **Minotaur, Maze, Robot**.

Game_engine::Game_engine(int n, int m2, int items_number)

Kod ove klase, konstruktor je dosta bitan. On kreira lavirint pozivom maze konstruktora, postavlja poziciju robota i minotaura koristeći set_position metodu, i podešava **seme (seed)** generatora pseudorandom brojeva.

void Game_engine::display_start()

Ovom metodom prikazujemo bitne informacije pre prikaza table lavirinta. Prikazuje se koji predmet je trenutno aktivan, i da li mu je vreme delovanja isteklo, a ukoliko nema nijedan, prikazuje se “None”. Takođe ovde proveravamo efekat magle rata, da bismo pozvali njen efekat umesto standardne render.display() metode

void Game_engine::run()

Glavna metoda klase. Simuliran tok igre. Sve prethodno napravljeno je objedinjeno ovom metodom. Tok igre je **while(true)** petlja, koja ceka interakciju sa korisnikom. Prvo se prikazuje lavirint. Onda se pozove funkcija robota **play**, i posle toga funkcija minotaura **play.** Ovde proveravamo kraj partije (Q) ili pobedu robota. Takođe, ovde se proverava efekat 4. predmeta, štita, jer ukoliko je aktivan, minotaur ne može da pojede robota.

Item

Ova klasa je čista virtualna klasa, zbog korišćenja ključne reči virtual u deklarisanju njenih metoda. Predstavlja osnovu klasu predmeta, koju kasnije nasleđuju konkretni predmeti. Klasa ima kao atribute **duration, r, c**. Oni predstavljaju dužinu trajanja efekta predmeta i poziciju u matrici. Njene virtualne metode koje moraju da se “override” - uju su **get_type** i **effect.** Efekat u svakoj klasi naslednici smanjuje trajanje za 1.

virtual std::string get_type() = 0

Ova funkcija vraća string, i za svaku klasu naslednicu je analogna. Vraća string koji predstavlja ime klase, da bismo znali koji objekat je konkretno predmet. Recimo:

```
std::string Fog_of_war::get_type() {
return "Fog_of_war";
}
```

Shield

bool Shield::effect(Maze& maze, int r, int c)

Efekat štita je postavljanje minotaura na poziciju r, c. Vraća true.

Hammer

bool Hammer::effect(Maze& maze, int r, int c)

Efekat predstavlja uslov prolaska robota. Vraća true ili false u zavisnosti da li robot može da prođe. Zbog ostale implementacije, vraća false ako robot može da prodje, odnosno true ukoliko ne može da prodje (polje je minotaur ili ulaz)

Sword

bool Sword::effect(Maze& maze, int r, int c)

Dozvoljava robotu da pređe preko minotaura, jer će ga tako ubiti. Vraća uslov da li robot može da stane na polje r, c ili ne može.

Fog_of_war

bool Fog_of_war::effect(Maze& maze, int r, int c)

Ova funkcija vraća true uvek. Prikazuje podmatricu 3x3 u odnosu na robota, to jest polje r, c.

MazeGame

Main

Struktura main fajla je provera da li je sve u redu pre pokretanja programa. To ukljucuje sledeće provere.

- **Provera parametara:** Ukoliko program ne primi 3 ulazna parametra, ne možemo da kreiramo normalan lavirint. Ova greška ima izlazni kod 1. U kodu proveravamo **argc** vrednost, koja mora da bude 4, zato što je prvi parametar naziv programa koji treba da se izvrši, i onda slede jos 3 broja.
- **Validacija parametara:** Kada pretvorimo parametre u cele brojeve, moraju da budu u određenom opsegu. Za dimenzije, minimalna dužina je 16, a broj

predmeta je minimalno 4. Takođe postoji i gornja granica koja je 300. Izlazni kod ove greske je 2

- **Broj predmeta:** Minimalni broj predmeta je 4. Međutim, maksimalni broj zavisi od dimenzija lavirinta. Izračunavamo broj dostupnih polja u lavirintu, i ukoliko je broj predmeta veći od ovog broja, to je greška. Izlazni kod je 3.

- **Prekoračenje stacka:** Pošto naš lavirint zauzima memoriju na heapu, i kasnije prilikom poziva funkcija koje se kreiraju na stacku i zauzimaju memoriju, ograničili smo ukupnu veličinu memorije koju lavirint može da zauzme. To je 10 000. Izlazni kod ove greške je 4.

Kada smo utvrdili da nema grešaka, pokrećemo igricu tako što napravimo objekat Game_engine, koji u svom konstrukturu pripremi sve potrebno za nastavak, i posle kreiranja objekta pozovemo metodu run().

Nakon završene metode run, znači da je došlo do kraja partije, pa pozivamo save_game da bismo zabeležili poslednju sliku igrice.

Pokretanje

Kako vršimo pokretanje našeg programa? Moramo kroz terminal da se pozicioniramo u direktorijum gde se nalazi MazeGame.exe, binarni izvršni fajl koji smo generisali buildovanjem projekta. Zatim kucamo u terminal sledeću komandu:

./MazeGame.exe N M X

- N predstavlja broj redova u lavirintu
- M predstavlja broj kolona u lavirintu
- X predstavlja broj specijalnih predmeta

Testiranje

Program je testiran na greške prilikom pokretanja pre samog generisanja lavirinta. Pošto naš program izbacuje greške u vidu izlaznog koda, koristićemo komandu

echo $?

da bismo videli izlazni kod. Najbolje je koristiti neki standardni terminal kao što je bash, jer kod nekih ova komanda neće lepo interpretirati izlazni kod.

Svi testovi su dostavljeni u dodatnom dokumentu **testovi.doc,** i tamo je prikazano kako se program ponaša tokom različitih unosa i scenarija.

./MazeGame.exe 22 7 8

Error: Rows and columns must be >= 15, and the number of items must be >= 3.

./MazeGame.exe 40 20

Error: Wrong number of parameters

./MazeGame.exe 30 17 5000

Error: The number of items can't fit in given dimension

./MazeGame.exe 150 160 50

Error: Stack overflow, choose smaller matrix

Item: None

#######U############

####.......#.P.P..P#

#........#...#.....#

#P##......##...#...#

#....##.#.#.##..P.P#

#...###...#........#

##....#..P.#.P#P...#

#P...P...#P..#.#...#

#P..#..#....#..P...#

#..P#..P....#.#P.P.#

##..#.#P....#..#.PP#

##.P......#.#..#P..#

#.....#...#...P...P#

#.##..#.#.......P.P#

#.##....#..###....##

#.#.#.####....P#PP.#

##.#.##.....P.#.#.##

#.....#.#..##...#..#

##....P#.#.#....PP.#

#.....**M**.**R**....P#...#

##...P..##....P..#.#

#.................#

#P.....#........#..#

##.#P#P#....P....P.#

##################I##

LOST

U ovoj situaciji, robot je skrenuo levo, i pošto je minotaur bio na korak do njega, pojeo ga je i igrač je izgubio.

U istoj situaciji je robot imao predmet, mač ili stit, desio bi se efekat predmeta i ili bi robot ubio minotaura, ili bi se samo odbranio.

Testiran je izlazak robota van granica table, ukoliko poseduje čekić

Testirano je jedenje predmeta od strane minotaura

Testiran je upis u datoteku maze.txt

Napredni objektno orijentisani koncepti

Nasleđivanje

Nasleđivanje je koncept gde jednu klasu možemo da koristimo u više drugih klasa, preuzimajući njene atribute, metode, konstruktore i slično. U našem projektu, ovaj koncept smo koristili kod klase predmet, jer smo iz te klase (koja se naziva **roditeljska klasa, bazna klasa**) izveli nove 4 klase koje koriste iste atribute (**klase naslednice, izvedene klase**)

class Fog_of_war : public Item

Ovako, klasa Fog_of_war nasleđuje klasu Item. Ključna reč public znači da će izvedena klasa videti sve atribute iz roditeljske klase takvi kakvi jesu, osim private atribute (public ostaje public i protected ostaje protected). Analogno smo koristili public nasleđivanje i za ostala 3 predmeta

Polimorfizam i virtuelne funkcije

Polimorfizam je kada ponašanje nekih metoda objekata zavisi od konteksta. Jedna metoda može da ima isto ime ali da prima različite parametre i tako radi drugačije stvari. U našem projektu smo koristili polimorfizam kod nasleđivanja, gde smo override-ovali funkcije effect i get_type za svaki predmet

std::string get_type() override;

Za svaku izvedenu klasu iz Item, ova funkcija radi drugačiju stvar. To je takođe slučaj i sa effect funkcijom.

Virtuelne funkcije su članovi klase koji se mogu prepisivati**(overriding)** u izvedenim klasama. Kada koristite virtuelne funkcije, ponašanje funkcije se određuje u **runtime-u** (tokom izvršavanja programa), a ne u **compile-time-u** (tokom kompajliranja). U našem programo smo koristili to kod klase Item i njenih klasa naslednica.

virtual std::string get_type() = 0;

Ovako definisana virtuelna funkcija je čista virtuelna funkcija, koja sprečava instanciranje cele klase. Time smo zapravo stvorili apstraktnu klasu, pošto nam nije potrebno njeno instanciranje, već njene izvedene klase.

Preklapanje operatora

Preklapanje operatora je koncept u C++ programiranju koji omogućava definisanje ili redefinisanje ponašanja standardnih operatora (poput +, -, *, <<, >>) za korisnički definisane tipove (npr. klase). Mi smo ga koristili u klasi **Maze** gde preklopljen operator pristupa

char*& operator[](int row);

Ovim je postignuto da operator **[] []** pristupa odredjenom elementu u matrici board, koja zapravo predstavlja lavirint.

Typedef

Ovo je princip koji može da nam skrati pisanje koda, pojednostavi neke stvari, i pomogne u bržem kucanju. Omogućava da sami napravimo tip podatka, ili damo

skraćenicu za već postojeći (složen) tip. U projektu smo ga koristili kod ugrađenog tipa podatka par

typedef std::pair<int, int> pii

Izuzeci

Izuzeci su mehanizam za obradu i otkrivanje grešaka u programu, bez da do te greške zapravo dođe i time uništi rad programa. U našem programu smo koristili izuzetak kod preklopljenog operatora pristupa, i to **out_of_range**

```
if (row < 0 || row >= n) {
throw std::out_of_range("Row index out of bounds");
}
```

Zaključak i nedostaci

Vreme generisanja lavirinta

Prilikom testiranja, uvideli smo da vreme za generisanje nije jednako, što je i očekivano. Kako veličina lavirinta raste, tako raste i vreme generisanja.

./MazeGame.exe 16 16 10

Time to generate maze: 0.0004258 seconds

./MazeGame.exe 100 100 5000

Time to generate maze: 0.0150685 seconds

Vidimo da je vreme oko 25 puta veće za veliki lavirint. Međutim, to nije problem s obzirom da je vreme ispod 1 desetinke, pa se ni ne primeti.

Inventar

Što se tiče ostalih nedostataka, možemo da primetimo da ukoliko bi vreme trajanja efekta bila duža, mogli bismo da pokupimo nekoliko predmeta u isto vreme. U našem slučaju, predmeti se samo preklope, ažurirajući vreme trajanja na 3 poteza i

postavljanjem novog itema. Ako bismo predmete čuvali kao nekakvu listu, mogli bismo da imamo više predmeta odjednom, i tako primenimo nekoliko efekata u isto vreme.

Čekić

Specijalni efekat čekića dozvoljava nam da prolazimo kroz zid. Kada robot prođe kroz zid, na mesto gde je bio zid se ponovo stvori zid, i ukoliko bi robot uspeo da uđe u “prostoriju” okruženu zidom, a specijalni efekat mu istekne, ne bi mogao da izađe.

.#..#....

....###..

...#.R.#

...#..##.

..####...