UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
NOVI SAD

ISPITNI RAD

Kandidat: **Aleksa Ćurčić**
Broj indeksa: **SV70/2023**

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

Mentor rada: **Dušan Stojković**

Uvod

Projekat ***Robot u lavirintu u Knososu*** realizovan je kao individualni zadatak u okviru kursa *Objektno orijentisano programiranje 2 (OOP2)*. Igra je inspirisana mitovima o lavirintu na Kritu, koji je, prema legendi, izgradio Dedal i u kojem je boravilo mitsko čudovište Minotaur. Osnovni cilj projekta je kroz objektno orijentisani dizajn simulirati interaktivnu igru u kojoj korisnik upravlja robotom koji pokušava da pronađe izlaz iz lavirinta izbegavajući Minotaura i koristeći predmete sa posebnim efektima.

Opis igre

Igra se odvija u *lavirintu definisanom kao matrica*, čije se dimenzije i broj predmeta zadaju kao argumenti komandne linije. Lavirint se dinamički generiše pri pokretanju programa. U njemu se nalaze zidovi, prolazi, ulaz, izlaz, Robot kojim upravlja igrač, Minotaur koji se kreće samostalno, i specijalni predmeti koji utiču na tok igre.

Pravila i tok igre

Struktura lavirinta:

- Zidovi ('#') grade spoljašnji okvir i unutrašnje prepreke.
- Ulaz ('E') je nasumično postavljen u prvom redu.
- Robot ('R') se postavlja ispod ulaza.
- Izlaz ('X') je nasumično postavljen u poslednjem redu.
- Prolazi ('.') omogućavaju kretanje.
- Minotaur ('M') i predmeti ('?') se postavljaju nasumično, uz poštovanje uslova prolaznosti.

Igru igra jedan igrač tako što upravlja Robota.

Kretanje robota

Igrač unosi komande za kretanje Robota gore, dole, lijevo ili desno, ne dijagonalno, koje odgovaraju klasičnoj kombinaciji karaktera W, A, S, D. Robot ne može proći kroz zidove, osim ako ne posjeduje poseban predmet Čekić.

Ponašanje Minotaura

Kreće se nasumično, takođe u pravcima gore, dole, lijevo ili desno, ne dijagonalno i ne kroz zidove. Poseban slučaj je ako se u trenutku njegovog računanja sledećeg poteza Robot nalazi direktno pored njega, tada se neće kretati nasumično već pravo na Robota. Takođe uništava specijalne predmete ako stane na njih.

Predmeti i efekti

- Magla rata — smanjuje vidljivost (3x3 podmatrica).
- Mač — omogućava uništavanje Minotaura.
- Štit — štiti od napada Minotaura i blokira njegovo kretanje 2 poteza.
- Čekić — omogućava prolazak kroz zid.

Svi efekti traju tri poteza.

Završetak igre

Postoje samo tri načina da se igra završi:

- Robot stigne na izlaz.
- Minotaur pojede robota.
- Korisnik prekine igru komandom (‘Q’).

Kada se igra završi generisaće se fajl „output.txt“ u kome će se nalaziti rečenica koja objašnjava način na koji je igra završena, kao i konačno stanje lavirinta u trenutku završetka igre.

Strukura projekta

Struktura datoteka

Projekat je podjeljen u više datoteka radi bolje organizacije i veće čitljivosti, i to na header i source datoteke. U header datotekama se u potpunosti nalaze deklaracije svih klasa, metoda, članova i pomoćnih struktura.

- Game.h
- Item.h
- FileManager.h
- Helper.h
- Maze.h
- MazeGenerator.h

U source datotekama nalaze se definicije gore pomenutih komponenti. Spisak source datoteka je sličan spisku header datoteka:

- Game.cpp
- FileManager.cpp
- Maze.cpp
- MazeGenerator.cpp
- Main.cpp

Pregled klasa

Position

Koristi se za elegantan način pristupa određenoj lokaciji u lavirintu, a takođe sadrži i posebne preklopljene operatore za dodavanje Position i Direction radi lakšeg rada sa pomjeranjem entiteta i generisanjem lavirinta.

Od polja sadrži:

- int x,
- int y,

koji ukazuju na koordinatu u matrici lavirinta.

Konstruktori

- Konstruktor nema, već koristi Struct aggregate inicijalizaciju

Operatori

bool operator==(const Position& other) gdje se vrši provjera da li su polja x i y jednaki među dva objekta.

Position& operator+=(const Direction& dir) koji dodaje vrijednost direkcije na poziciju.

Position operator+(const Position& other) koji vraća novi Position objekat sa novim sabranim vrijednostima.

Game namespace

Odgovoran je za izvršavanje igre. U njoj se nalazi glavna petlja u kome se pozivaju adekvatne funkcije za iscrtavanje jednog frejma igre, funkcija za prijem unosa korisnika, kao i funkcije odgovorne za ažuriranje i kreiranje sledećeg frejma igre.

Od članova sadrži:

- **static Maze** s_maze,
- **static Robot s_**robot,
- **static Minotaur s_**minotaur

Inicijalizacija

Prima argumente sa komandne linije i prosljedjuje parser funkciji handleArguments koji vraća Settings objekat i ta podešavanja šalje generatoru. Potom inicijalizuje statičke članove robot, minotaur, maze.

Metode

- Void run() - glavna main petlja programa koja radi sve dok se ne ispune zahtjevi za izlazak iz aplijacije, ispisuje stanje lavirinta nakon svakog poteza, poziva na robotov/minotaurov potez.
- Bool Running() - provjerava da li se ispunio zahtjev za gašenje aplikacije (robot izašao iz lavirinta, robot umro od strane minotaura, korisnik pritisnuo q na tastaturi).
- Void Exit() - najprije ispisuje posljednje stanje lavirinta, sačuvava to stanje u fajl i gasi program.
- Void RobotTurn() - prvo uzima unos sa tastature i pretvara ga u direkciju pozivom funkcije directionFromChar koja opciono vraća direkciju ukoliko je unos ispravan. Potom računa novu poziciju robota i iskorošćava aktivni predmet koji robot sadrži. Predmet se koristi prije robotovog poteza da bi obezbjedila da se na novoj poziciji ili uništi zid, ili napadne/odbrani od monitaura kako bi obezbjedila robotu siguran prolaz na novu poziciju.
- Void MinotaurTurn() - provjerava da li je minotaur u mogućnosti da se kreće (odnosno da li je nokautiran, ili mrtav). Ukoliko nije u mogućnosti da se kreće, tok funkcije se prekida i vraća se na glavnu petlju. Ukoliko je minotaur u mogućnosti da napadne, onda se poziva funkcija minotaur.move() koja upravlja kretanjem minotaura i obnavljanjem stanja lavirinta nakon njegovog poteza. da efekat, ako postoji, modifikuje destinacionu ćeliju, tako da Robot nesmetano može na nju da pređe.

Varijable članice

- Static Maze s_maze
- Static Robot s_robot
- static Minotaur s_minotaur

  Varijable članice su statičke kako bi se obezbjedio pristup objektima kroz cijeli program. Razlog tome je da bi se izbjeglo skladištenje referenci (neki vid asocijacije),

proslijedjivanje referenci kroz konstruktor, I slično. Pošto je projekat relativno jednostavan (samo par jedinstvenih objekata) onda smatram da je ovo relativno najjednostavniji I najelegantniji pristup.

Bazna klasa Item

U kojoj se navode osnovna polja njenih nasljednica, kao i virtualne funkcije koje će nasljednice implementirati.

Što se tiče polja, sadrži:

- **int** m_duration
- **Type** m_type

Metode

Prisutne su

- **bool isBroken() const** - provjerava da li je predmet validan za korišćenje,
- **int duration() const** – vraca broj preostalih poteza efekta
- **void reduceDuration() -** smanjuje broj poteza

Od virtualnih imamo

- **void use(Position pos)** – čista virtualna funkcija koju će sve klase nasljednice da implementiraju, i ona je suštinska metoda ove klase. Proslijedjuje se pozicija na kojoj će se izvršiti efekat.
- **Std::string_view getStr()** - čista virtuelna funkcija koja vraća string reprezentaciju predmeta

Klase nasljednice

Implementiraju **use(Position pos)** i **getStr()** iz njihove bazne klase Item. Ideja za predmet je vrlo jednostavna, ako je aktivan treba da modifikuje određenu ćeliju. Naravno ovo ne važi za efekat Magle čiji je jedini zadatak iscrtavanje.

Hammer

Hammer implementira **use(Position pos)** tako što najprije provjeri da li pozicija ukazuje na zid i da li je pozicija unutar dozvoljenih granica lavirinta. Ako je u pitanju zid samo će ga „srušiti“ tako što će zamjeniti zid sa poljem prolaza.

Shield

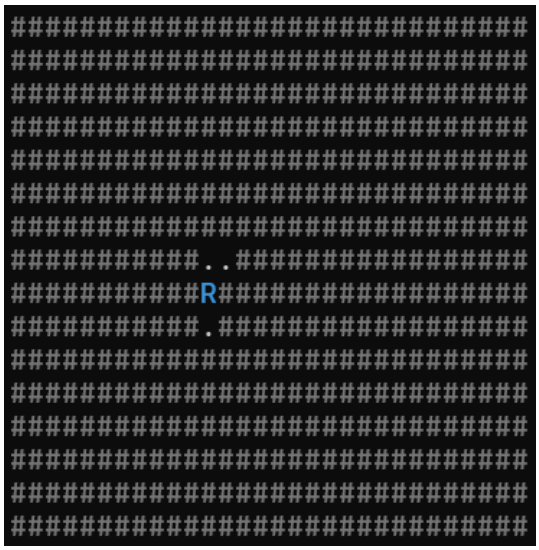
Shield implementira slično, međutim sada provjerava da li proslijedjena pozicija ukazuje na minotaura. Ukoliko pokazuje na minotaura, onda će ga nokautirati(zalediti 2 poteza) kako bi obezbjedila robotu da se udalji od minotaura i pobjegne sa scene.

Word

Sword koristi sličnu ideju kao i prethodna dva. U slučaju da je lokacija zapravo destinacija Minotaura, Minotaur umire.

FogOfWar

Ovo je jedini efekat koji odstupa od uobičajenog šablona rada efekata. Koristi se u preklopljenom operatoru operator<< kod maze klase tako što dinamičkim kastom provjeravamo da li robot ima fog kao aktivan predmet. Ukoliko je to slučaj onda će modifikovati prikaz lavirinta da prikazuje samo matricu 3x3 oko robota.



Lavirint dok je efekat Magle aktivan

Klasa MazeGenerator

Glavni zadatak ove klase, kao što joj i ime sugeriše, jeste kreiranje samog lavirinta na osnovu zadatih podešavanja. Zbog toga što lavirint „ne zna“ za entitete, već samo za svoje postojanje, onda ona neće postaviti samog robota i minotaura već se to ostavlja za korisnikovu implementaciju. U MazeData struktu je ponudjena matrica, kao i pozicija ulaza i izlaza i vrijeme potrebno za generisanje.

Konstruktor

MazeGenerator(Settings settings) prima podešavanja, i inicijalno popunjava matricu sa zidovima.

Metode

- **MazeData generate()** - glavna metoda za generisanje lavirinta koja vraća podatke vezane za lavirint i vrijeme generisanja. Radi koristeći backtracking algoritam implementiran sa stekom. Princip je veoma jednostavan – zadamo početnu poziciju odakle algoritam kreće (to će u našem slučaju biti pozicija ispod ulaza u lavirint) i odatle ulazimo u petlju koja radi sve dok ima nešto na steku. Skidamo zadnju poziciju sa steka i pozivamo funkciju randomDirection koja nam za trenutnu poziciju vraća random direkciju u kojem će se nastaviti generacija lavirinta. Ukoliko postoji mogućnost za trenutnu poziciju da se istražuje dalje, poziva se funkcija carvePath koja sa trenutne pozicije u zadatom smijeru obiljezava polja kao prolaze. Potom se na stek puša nova pozicija i nastavlja se dalje sa

petljom. Ukoliko se desilo da na trenutnoj poziciji nema više mogućnosti da se istražuje dalje, onda se vraća nazad kroz stek na prethodnu poziciju da se ona istraži.

Postoje i brojne pomoćne metode koje se koriste u procesu generisanja lavirinta:

- **Void carvePath(Position pos, Direction dir) –** na zadatoj poziciji u smijeru direkcije zabilježi polja kao prolaze
- **std::optional<Direction> randomDirection(Position pos) –** za zadatu poziciju vraća random direkciju za nastavak generisanja lavirinta. Ukoliko nema validne direkcije onda vraća std::nullopt.
- **Bool isValidDirection(Position pos) –** provjerava da li je pozicija unutar granica lavirinta i da li nova pozicija ukazuje na zid. Ukoliko ne ukazuje na zid – to znači da je ta pozicija već istražena i stoga nevalidna za daljnje istraživanje.
- **Void placeItems() -** funkcija koja nasumično markira polja kao predmete. Ne instancira predmete, već radi samo označavanje.
- **Void carveAdditionalPassages() -** funkcija koja čisti zadnji red u lavirintu kako bi obezbjedio više od jednog izlaza iz lavirinta, i pored toga nasumično bira poziciju na kojem će počistiti par zidova kako bi stvorio više prolaza. Cijela uloga funkcije je da obezbjedi lakše kretanje rovota kroz lavirint.

FileManager

Koja je zadužena za upisivanje u fajl konačnog stanja igre kao i vremena generisanja lavirinta.

U konstruktoru prima ime fajla i odatle inicijalizuje svoj file stream i čuva ime fajla sa kojim radi.

Metode

- **Void save(const T& obj) –** templejtovana funkcija koja prima objekat koji će se čuvati. Objekat koji se proslijedjuje mora da overloaduje operator<< za rad s fstream-om.
- Std::string_view fileName() - metoda koja samo vraća ime fajla sa kojim menadžer radi.

Maze

Kao što samo ime sugeriše ona predstavlja implementaciju samog lavirinta. Sadrži metode koje daju bilo koju informaciju koju želimo da znamo o trenutnom stanju lavirinta.

Od polja sadrži:

- **MazeData** m_data – sadrži sam vektor vektora, koji predstavlja matricu, poziciju ulaza i izlaza kao i vrijeme generacije.

Konstruktior

Posjeduje samo parametrizovan konstruktor

Maze(MazeData data),

koji podešava polje m_data.

Metode

- **Bool isWalkable(Position pos) –** vraća da li je pozicija unutar granica i provjerava da li je pozicija validna za hodanje entiteta po njoj
- **void swapCells(Position, Position)** – mijenja polja matrice. Takodje vrši provjeru da li je jedna od ćelija entitet, te stoga ukoliko entitet nagazi na predmet/izlaz da može da zamjeni tu ćeliju sa prolazom.
- **Void updateCell(Position, Cell**) – na zadatoj poziciji markiraj polje sa zadatim poljem
- **bool isWithinBounds(Position)** – provjerava da li je pozicija unutar granica lavirinta

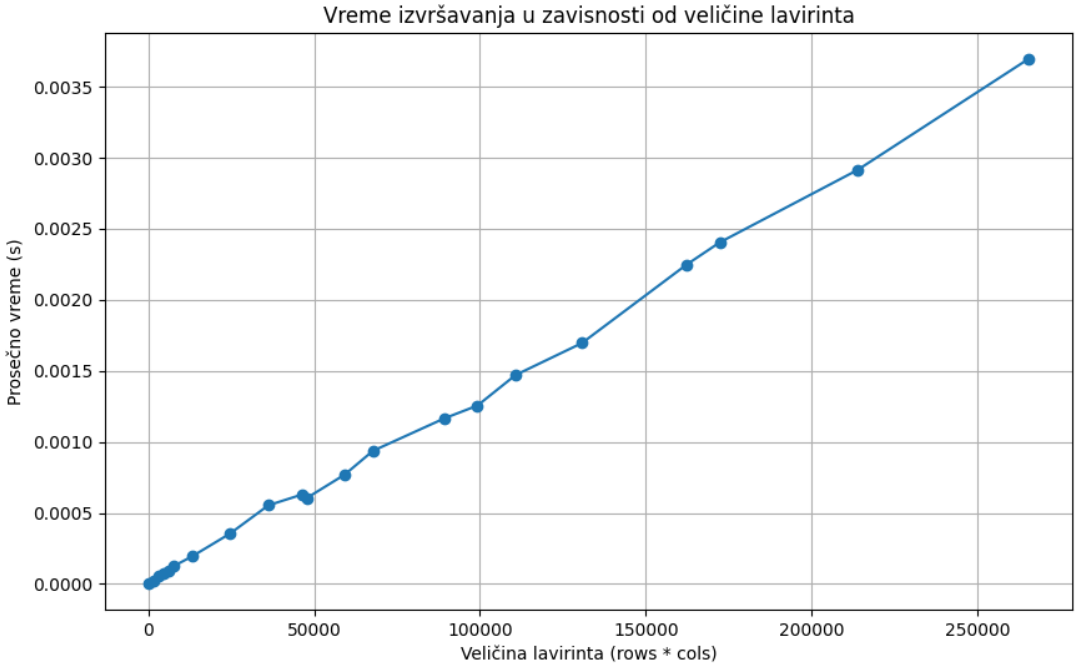
Spisak korisničkih enumeracija

Pored gore pomenutih klasa, postoje i korisničke enumeracije koje se koriste kao slikovite konstante.

- **Item::Type** – služi da slikovito navede koji su to specijalni efekti mogući.
- **Cell** – slićno EffectType-u, koristi se da navede sve moguće tipove pojedinačne ćelije u lavirintu.

Analiza vremena

Zavisnost vremena generisanja od dimenzija lavirinta



Na grafikonu su prikazana vremena generisanja lavirinta u zavisnosti od zadatih dimenzija lavirinta. Broj predmeta ostavljen je na podrazumjevanoj vrijednosti, to jest broj predmeta je fiksiran na 3.

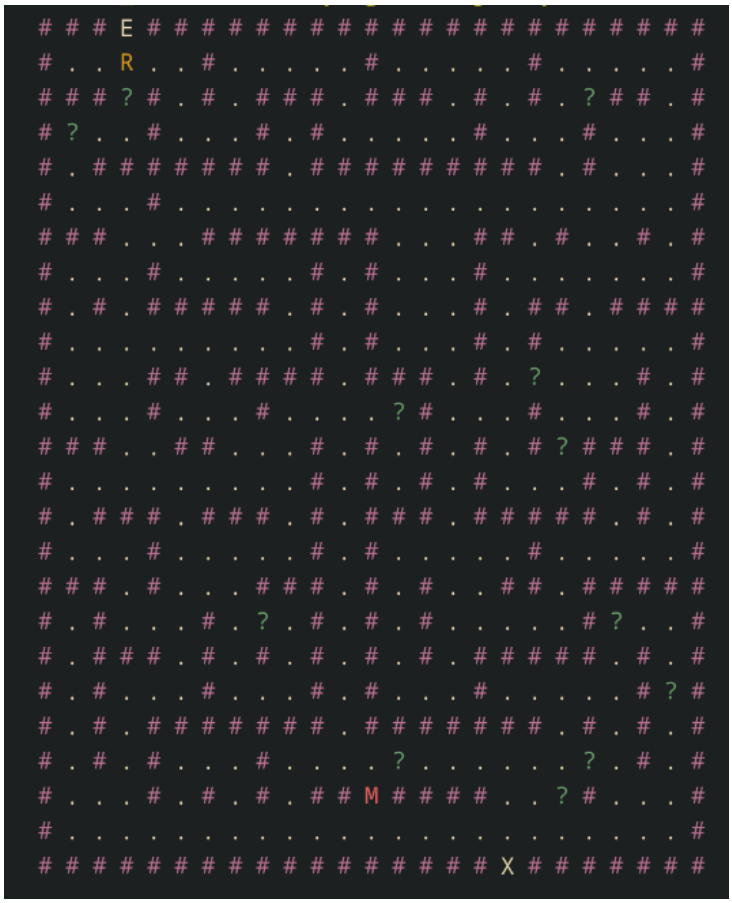
Dakle algoritam ima neku linearnu zavisnost od dimenzije, to jest od broj_redova * broj_kolona.

Testiranje

Što se tiče testiranja same igre, neki klasični primjeri jesu provjera ispravnosti programa kada se unesi pogresni argumenti. Tu spada pogrešan broja argumenata, sam tip argumenta, recimo ako proslijedimo string, a i sama ograničenja argumenata, recimo da broj kolona mora biti bar 16. Najprije ću pokazati primjere ispravne upotrebe programa, to jest kada su svi zahtjevi za argumenti ispunjeni.

Dakle sve što treba uraditi u konzoli jeste odabrati izvršavanje programa i proslijediti parametre odvojene razmakom, i to redosljedom <broj redova> <broj kolona> <broj predmeta>

Primjeri generisanih mapa:



Broj redova je 25, broj kolona 25, a broj predmeta 12

Izlazni rezultati

Za kraj preostaje da se provjeri izlazni fajl “build/output.txt”, koji se generiše nakon svakog pokretanja igre, u kome se nalazi zapis posljednjeg stanja igre, kao i vrijeme potrebno da se generise lavirint.