Mlin

Nikola Vučković

15. lipnja 2020.

Sadržaj

1 Uvod

Mlin je zabavna i jednostavna igra nastala još u Srednjem vijeku. Za igru se koristi igrača ploča i 9 figurica. Cilj je postaviti tri svoje figure na susjedne točke povezane linijom. Tako formirana „trojka" se naziva mlin. Igrač koji formira „mlin", uklanja jednu protivničku figuru po vlastitom izboru. To jedino ne smije biti niti jedna figura iz ranije formiranog protivničkog „mlina". U daljnjem tijeku igre, formirani „mlin" se može rasformirati jedino pomicanjem jedne od tri njegove figure. U nekom sljedećem potezu, isti igrač ga može ponovno formirati, vraćanjem iste figure na staro mjesto. Pri tome ponovno osvaja jednu protivničku figuru. U slučaju da su sve protivničke figure u sastavu „mlina", uklanja se bilo koja od njih. Pobjednik je igrač koji osvoji sedam protivničkih figura, odnosno, ostavi protivnika sa samo dvije figure ili ostavi protivnika bez mogućeg sljedećeg poteza.

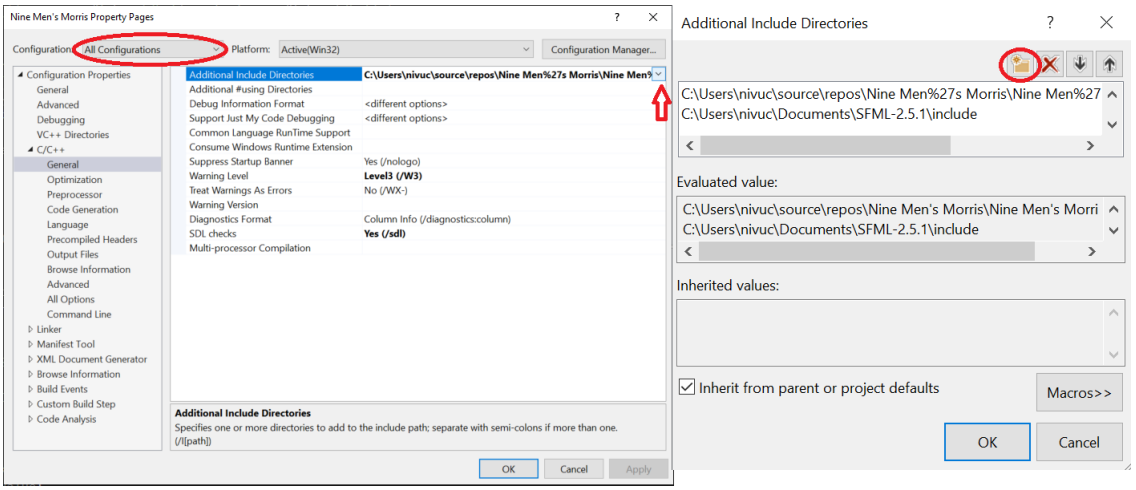
Tijek igre sastoji se od 3 faze:

1. Postavljanje figura
    - Na samom početku igre, igrača ploča je prazna. Prvo crni, a zatim naizmjenično, igrači postavljaju po jednu figuricu na slobodnu, presječnu točku. U ovoj fazi igre, postavljaju se uvijek nove figure, a one na ploči se još uvijek ne smiju pomicati. Ako igrač u ovoj fazi formira „mlin", odmah uklanja jednu protivničku figuru s ploče. Kad se nakon 9 poteza (9 figura) , sve figure uvedu u igru, prelazi se na sljedeću fazu.
2. Pomicanje figura na susjedna polja
    - Igrači naizmjenično pomiču po jednu svoju figuru, duž linija, do susjedne točke. U ovoj fazi, može se pomaknuti jedna figura iz formiranog „mlina", a u sljedećem potezu vratiti, formirati ponovo i iznova ukloniti protivničku figuru. Za rasformiranje formiranog „mlina", koristi se termin „otvaranje", a za ponovno njeno formiranje, termin „zatvaranje". Čim igrač ostane na tri figure, samo on ulazi u treću fazu igre.
3. „Flying" faza
    - Igrač koji ostane s tri figure, ima pravo bilo koju od njih prebaciti na bilo koje slobodno polje. Ako u međuvremenu i protivnik ostane sa tri figure, i on tog trenutka ulazi u 3. fazu igre. Znači, i njegove figure sada mogu skakati od polja do polja, bez obzira na linije spajanja. Partija „mlina" se u drugoj fazi igre može i završiti „blokiranjem". Ako bilo tko, na svom potezu, nema što odigrati - gubi partiju.

2 Projekt

Zadatak je napravljen u razvojnom okruženju *Microsoft Visual Studio 2019* korištenjem programskog jezika *C++* i biblioteke *SFML 2.5.1* za iscrtavanje i zvukove. Kako bi se

projekt uspješno kompilirao potrebno je povezati *SFML* biblioteku s projektom u *Visual Studio-u*. Detaljne upute za instalaciju nalaze se na https://www.sfml-dev.org/tutorials/2.5/start-vc.php dok se korištena verzija biblioteke *SFML* može preuzeti s https://www.sfml-dev.org/download/sfml/2.5.1/ (za ovaj zadatak korištena je verzija za *Visual C++ 15* i 32-bitne platforme). Unutar projekta potrebno je još dodati folder gdje se nalaze headeri koji se koriste u projektu.



Slika 1: Potrebno je dodati /. . . /Nine Men's Morris/Nine Men's Morris/Include folder

Također, igru je moguće pokrenuti u **debug** ili **release** mode-u koji se nalaze unutar pripadnih foldera.

3 Arhitektura rješenja

Primarna klasa koja je „kostur" aplikacije je **GameEngine**. Ona će obavljati sljedeće poslove:

- Pokretanje glavnog prozora
  - Pozivom konstruktora klase kreira se prozor veličine 640×800, gdje je također podešen *antialiasing* tako da zaobljeni objekti nemaju grube rubove.
- Upravljanje **GameState** stackom
  - Klasa **State** (koju ćemo malo kasnije opisati) predstavljat će nam stanje na prozoru. **GameEngine** je zadužen za ispravno prebacivanje između **GameState-ova** i njihovo ažuriranje.
- Ažuriranje prozora
  - Svakom iteracijom **GameEngine** će registrirati nove evente te ih proslijediti trenutno postavljenim **GameState-ovima**, ažurirat će i iscrtati na ekran trenutno stanje objekata unutar postavljenih **GameState-ova**.

Parametrizirana klasa **ResourceHolder** je glavni spremnik za učitane resurse koji mogu biti tipa: *sf::Texture*, *sf::Font* i *sf::SoundBuffer*. Funkcijom *getResource* učitavamo traženi resurs.

Već spomenuti **GameState** je bazna klasa iz koje će biti izvedeni svi ostali stateovi koji će biti korišteni u programu. Klasa sadrži sljedeće funkcije i metode:

- GameState konstruktor
  - Konstruktor prima klasu **Context** koja sadrži kazaljke na spremnike za resurse i **GameEngine**. Tako će sve izvedene klase i njihovi objekti imati pristup objektima čije adrese su pohranjene u klasi **Context**.
- *render, update, input*
  - Funkcije će se pozvati na svim pripadnim **GameObject** objektima (također će biti objašnjeno kasnije).
  - Povratna vrijednost reći će **GameEngine-u** treba li pozvati funkciju za sljedeći **GameState** u stacku. Ako funkcija klase na vrhu stacka vrati *true*, **GameEngine** će nastaviti pozivati funkciju za ostale stateove unutar stacka. Tako ćemo moći postići npr. osjećaj dubine kod *render* funkcije jer će se state na vrhu stacka iscrtavati preko state-ova ispod njega.
- Spremnik s **GameObject-ima**
  - Spremnik koji sadrži sve objekte koji se nalaze unutar scene (state-a).

Klasa **GameObject** je apstraktna bazna klasa za objekte koji će se nalaziti unutar pripadne scene. Iz nje će biti izvedene klasa **Node** i klasa **Button** koja će služiti kao *UI* komponenta. Unutar klase imamo:

- *render, update, input*
  - Funkcije koje će pozvati state kojoj objekt pripada.
- *onFocus, whileFocused, onFocusExit, onClick*
  - Metode koje će se izvršiti ukoliko se dogodio određeni *event* s mišem.

Klasa **Button** izvedena je iz klase **GameObject** i bit će upravo gumb. Svaki **Button** sadrži svoj *label*, tj. tekst koji će biti ispisan unutar gumba i *onClickFunc*, tj. funkciju koja će biti izvršena ako gumb bude pritisnut. Lako je proširiti klasu tako da je moguće pozivati zadane funkcije ne samo kada je gumb kliknut, nego kada je i npr. miš ušao unutar *collider-a* od gumba ili ako je izašao van *collider-a*.
Klasa **Board** je glavna klasa koja provjerava logiku igre. Svakim pozivom funkcije *update* ažuriramo stanje na ploči te postavljamo „ponašanje" **node-ova** ovisno o pripadnom stanju. Ukoliko je npr. „Player 1" u stanju „Placing" tada će se pritiskom lijevog gumba na mišu **node** provjeriti je li okupiran od strane nekog igrača i ako nije postaviti zastavicu da ga je okupirao igrač na potezu.

Klasa **Node** nam predstavlja „čvor" na ploči te je glavni objekt koji koristimo za pohranu informacija koje su potrebne za igru. Funkcijom *setOnClick* postavljamo ponašanje prilikom klika na node. Ovim pristupom lagano možemo implementirati i proširiti pravila igre.

Ploču reprezentiramo matricom susjedstva $M \in \mathbb{R}^{m\times 8}$, gdje je m broj čvorova u grafu. Na primjer, u verziji igre *Morabaraba* susjede od gornjeg lijevog čvora prikazat ćemo vektorom:

$$\big[-1, -1,\ 1,\ 3,\ 9, -1, -1, -1\big] \in \mathbb{R}^{1\times 8}.$$

i-ti element vektora nam govori ima li čvor susjeda u pripadnom smjeru, počevši od smjera sjevera (gore) pa u smjeru kazaljke na satu. Tako čvor u gornjem lijevom kutu (pripadni indeks 0) desno od sebe ima čvor s indeksom 3, ispod-desno od sebe čvor s indeksom 3 i ispod sebe čvor s indeksom 9. Ovakvom reprezentacijom ploče možemo lagano dodavati nove čvorove i veze među njima, tj. možemo jednostavno dodavati nove **Level-e**.

Klasa **AudioManager** se brine za učitavanje i puštanje zvukova. Metodama *toggleSounds* i *toggleBackgroundMusic* možemo podesiti jačinu zvuka na 0% ili 100% ovisno o prethodnom stanju.