

Haskell šah

Alesandro Žužić

23.06.2024.

Uvod

Ovaj projekt implementira klasičnu igru šaha koristeći Haskell i Gloss biblioteku za grafički prikaz. Cilj projekta je pružiti korisnicima funkcionalno i vizualno privlačno okruženje za igranje šaha, integrirajući osnovna pravila i logiku igre sa grafičkim sučeljem.

Projekt je strukturiran kako bi omogućio jednostavno igranje šaha uz mogućnost interakcije putem terminala. Osnovna funkcionalnost uključuje postavljanje i prikaz šahovske ploče, unos poteza od strane igrača, validaciju tih poteza te ažuriranje stanja igre. Gloss biblioteka se koristi za crtanje ploče i figura.

Projekt je organiziran u nekoliko modula, svaki sa specifičnom funkcionalnošću koja doprinosi cjelokupnoj igri šaha. Moduli su dizajnirani tako da omogućuju jednostavno održavanje i proširenje koda. Ova struktura omogućava centralizirano upravljanje šahovskom pločom i figurama, dok se interakcija s igračem odvija putem terminala.



Figure 1: Haskell Šah

Sadržaj

1. Uvod
2. Ključne Značajke
3. Ograničenja Projekta
4. Organizacija Projekta
 - Moduli Projekta
5. Struktura Direktorija
6. Moduli
 - Main
 - ChessPieces
 - ChessSprites
 - Chessboard
 - ChessLogic
 - PlayerInput
7. Funkcijsko programiranje u razvoju šahovske igre
8. Korišteni resursi

Ključne značajke:

1. **Grafički Prikaz:**
 - Koristi Gloss biblioteku za crtanje šahovske ploče i figura.
 - Ploča i figure se kombiniraju i prikazuju u prozoru aplikacije.
2. **Interakcija Igrača:**
 - Igrači mogu unositi poteze putem terminala.
 - Unos se obrađuje u zasebnoj dretvi kako bi se omogućilo paralelno izvršavanje grafičke simulacije i obrade unosa.
3. **Simulacija:**
 - Kontinuirano ažurira prikaz šahovske ploče na temelju poteza igrača.
 - Koristi `simulateIO` iz Gloss biblioteke za upravljanje simulacijom.
4. **Logika Igre:**
 - **Validacija Poteza Igrača:** Provjerava jesu li potezi igrača valjani prema osnovnim pravilima šaha.
 - **Praćenje Boje Igrača:** Ažurira i prati trenutnu boju igrača, osiguravajući da se igrači izmjenjuju u skladu s pravilima.

Ograničenja projekta

Iako projekt pruža osnovnu funkcionalnost igranja šaha, postoje određena ograničenja i funkcionalnosti koje nisu implementirane:

1. **Nedostatak Napredne Validacije Pravila:**
 - Projekt ne provjerava složena pravila šaha poput rošade, en passant hvatanja, i promocije pijuna.
2. **Nedostatak Grafičkog Korisničkog Sučelja (GUI):**
 - Interakcija igrača se odvija isključivo putem terminala. Nema implementiranih grafičkih elemenata za unos poteza putem miša ili drugih GUI kontrola.
3. **Nedostatak Provjere Šah-Mata i Pata:**
 - Projekt ne provjerava uvjete za šah-mat ili pat. Igrači moraju sami prepoznati kada je igra završena.
4. **Nedostatak Podešavanja i Opcija Igranja:**
 - Nema mogućnosti odabira različitih postavki igre poput vremenskog ograničenja, različitih početnih pozicija, ili igranja sa specifičnim pravilima.
5. **Nedostatak Spremljenih Partija:**
 - Projekt ne podržava spremanje i učitavanje partija. Svaka igra mora se odigrati u jednom sjedanju.
6. **Nedostatak Zvuka i Animacija:**
 - Projekt ne uključuje zvukove ili animacije za poteze figura, što može umanjiti iskustvo igranja.

7. Nedostatak AI Protivnika:

- Projekt ne podržava igru protiv računala. Nema implementiranog algoritma za umjetnu inteligenciju koja bi igrala poteze protiv ljudskog igrača.

Organizacija Projekta

Projekt je organiziran u nekoliko modula, svaki sa specifičnom funkcionalnošću koja doprinosi cjelokupnoj igri šaha. Ova modularna struktura olakšava održavanje i proširenje koda.

Moduli Projekta

1. Main.hs

- **Opis:** Glavni ulazni modul koji pokreće program.
- **Funkcionalnost:** Inicijalizira šahovsku ploču, boju igrača, pokreće petlju za unos putem terminala i pokreće grafičku simulaciju.

2. Chessboard.hs

- **Opis:** Modul koji definira šahovsku ploču i osnovne operacije na njoj.
- **Funkcionalnost:**
 - Inicijalizira početno stanje šahovske ploče.
 - Definira funkcije za crtanje šahovske ploče.
 - Pruža funkcije za dohvaćanje i provjeru stanja pojedinih polja na ploči.

3. ChessSprites.hs

- **Opis:** Modul koji se bavi crtanjem šahovskih figura kao spriteova.
- **Funkcionalnost:** Učitava slike figura i postavlja ih na odgovarajuća mjesta na ploči.

4. PlayerInput.hs

- **Opis:** Modul za rukovanje unosom igrača putem terminala.
- **Funkcionalnost:**
 - Pokreće petlju koja čeka unos igrača.
 - Validira unose igrača i ažurira stanje šahovske ploče.
 - Upravljanje izmjenom poteza između igrača.

5. ChessPieces.hs

- **Opis:** Modul koji definira šahovske figure i njihove karakteristike.
- **Funkcionalnost:**
 - Definira tipove podataka za figure i boje.
 - Pruža funkcije za rad s figurama, uključujući njihovu boju i vrstu.

6. ChessLogic.hs

- **Opis:** Modul za implementaciju logike igre.
- **Funkcionalnost:**
 - Provjerava validnost poteza.
 - Ažurira stanje ploče nakon valjanog poteza.
 - Upravlja izmjenom boje igrača.

Struktura Direktorija

```
.
|-- Main.hs
|-- Chessboard.hs
|-- ChessSprites.hs
|-- PlayerInput.hs
|-- ChessPieces.hs
|-- ChessLogic.hs
|-- sprites
    |-- white_king.bmp
    |-- white_queen.bmp
    |-- white_rook.bmp
    |-- white_bishop.bmp
    |-- white_knight.bmp
    |-- white_pawn.bmp
    |-- black_king.bmp
    |-- black_queen.bmp
    |-- black_rook.bmp
    |-- black_bishop.bmp
    |-- black_knight.bmp
    |-- black_pawn.bmp
```

Direktorij `sprites` sadrži BMP slike šahovskih figura koje se koriste za grafički prikaz ploče u igri.

Svaki modul ima specifičnu ulogu i zajedno omogućavaju funkcionalnost šahovske igre. Ova organizacija omogućava jednostavnije upravljanje i proširenje koda, čineći projekt preglednim i modularnim.

Moduli

Main

Ova datoteka implementira jednostavnu šahovsku igru koristeći Haskell i Gloss biblioteku za grafiku. Program uključuje:

1. **Inicijalizacija stanja:**
 - Kreira početno stanje šahovske ploče i postavlja početnu boju igrača na bijelu.
 - Koristi promjenjive reference (`IORef`) za držanje stanja ploče i trenutne boje igrača.
2. **Rukovanje unosom igrača:**
 - Pokreće petlju za unos naredbi igrača u zasebnoj niti koristeći `forkIO`. Ova petlja omogućava igračima da unose poteze putem terminala.
3. **Grafička simulacija:**
 - Pokreće grafičku simulaciju šahovske ploče koristeći Gloss. Simulacija kontinuirano osvježava prikaz ploče.
 - Kombinira vizualni prikaz šahovske ploče i figura kako bi se igračima omogućilo praćenje stanja igre.
4. **Ažuriranje stanja:**
 - Funkcija za ažuriranje stanja ploče koristi se za osvježavanje prikaza na temelju promjena unesenih putem terminala.

Ključne komponente

- **IORef:** Koristi se za čuvanje i modifikaciju stanja šahovske ploče i trenutne boje igrača.
- **Gloss:** Biblioteka za jednostavno stvaranje grafičkih aplikacija. Koristi se za crtanje šahovske ploče i figura.
- **Concurrency:** `forkIO` se koristi za pokretanje paralelne niti koja rukuje unosom igrača, omogućavajući da grafička simulacija i unos igrača rade istovremeno.

Proces izvršavanja

1. **Pokretanje programa:** Inicijalizira se početna šahovska ploča i boja igrača.
2. **Paralelno izvršavanje:** Pokreće se petlja za unos igrača u zasebnoj niti.
3. **Grafička simulacija:** Simulacija prikazuje šahovsku ploču i ažurira je na temelju poteza unesenih putem terminala.
4. **Interakcija:** Igrači unose poteze putem terminala, a promjene se prikazuju na grafičkoj ploči u stvarnom vremenu.

```
module Main where

-- Uvoz potrebnih modula
import Data.IORef (IORef, newIORef, readIORef) -- Uvoz IORef modula za promjenjive
-- reference u IO
import Graphics.Gloss (Display (InWindow), Picture, black, pictures) -- Uvoz Gloss
-- biblioteke za grafiku
import Graphics.Gloss.Interface.IO.Simulate (ViewPort, simulateIO) -- Uvoz simulacijskog
-- sučelja Gloss biblioteke za IO
import Control.Concurrent (forkIO) -- Uvoz modula za istovremeno izvršavanje

-- Uvoz modula projekta
import Chessboard (initialChessboard, drawChessboard, Chessboard) -- Uvoz funkcija i
-- tipova iz Chessboard modula
import ChessSprites (drawChessboardSprites) -- Uvoz funkcije za crtanje sprite-ova
-- šahovske ploče
import PlayerInput (terminalInputLoop) -- Uvoz funkcije za rukovanje unosom igrača
```

```

import ChessPieces -- Uvoz definicija i tipova vezanih uz šahovske figure

-- Glavna ulazna točka programa
main :: IO ()
main = do
    boardRef <- newIORef initialChessboard -- Kreiraj novi IORef koji sadrži početnu
    ↪ šahovsku ploču
    colorRef <- newIORef White -- Kreiraj novi IORef koji sadrži početnu boju igrača
    ↪ (Bijela)
    forkIO $ terminalInputLoop boardRef colorRef -- Pokreni petlju za unos s terminala
    ↪ istovremeno koristeći forkIO
    runProgram boardRef (InWindow "Chessboard" (500, 500) (100, 100)) -- Pokreni
    ↪ simulacijski program s danom referencom ploče i postavkama prikaza

-- Funkcija za pokretanje simulacijskog programa
runProgram :: IORef Chessboard -> Display -> IO ()
runProgram boardRef display =
    simulateIO display black 10 initialChessboard env2Pic (step boardRef) -- simulateIO
    ↪ pokreće Gloss simulaciju s početnim okruženjem, funkcijom renderiranja i funkcijom
    ↪ koraka ažuriranja

-- Funkcija za pretvaranje šahovske ploče u sliku
env2Pic :: Chessboard -> IO Picture
env2Pic board = do
    sprites <- drawChessboardSprites board -- Nacrtaj spriteove za šahovsku ploču
    let boardPic = drawChessboard -- Nacrtaj mrežu šahovske ploče
    return $ pictures [boardPic, sprites] -- Kombiniraj mrežu i spriteove u jednu sliku
    ↪ koristeći pictures

-- Funkcija za ažuriranje okruženja (šahovske ploče) na temelju IORef
step :: IORef Chessboard -> ViewPort -> Float -> Chessboard -> IO Chessboard
step boardRef _ _ = readIORef boardRef -- Pročitaj trenutno stanje šahovske ploče iz
    ↪ IORef i vrati ga

```

1. Module Declaration:

- module Main where: Definira glavni modul programa.

2. Imports:

- import Data.IORef (IORef, newIORef, readIORef): Uvozi funkcije i tipove za rad s promjenjivim referencama u IO monadi.
- import Graphics.Gloss (Display (InWindow), Picture, black, pictures): Uvozi dijelove Gloss biblioteke potrebne za grafiku, uključujući tipove za prikaz, slike i boje.
- import Graphics.Gloss.Interface.IO.Simulate (ViewPort, simulateIO): Uvozi Gloss funkcije za simulaciju s IO.
- import Control.Concurrent (forkIO): Uvozi funkciju za paralelno izvršavanje.
- import Chessboard (initialChessboard, drawChessboard, Chessboard): Uvozi funkcije i tipove za rad sa šahovskom pločom.
- import ChessSprites (drawChessboardSprites): Uvozi funkciju za crtanje šahovskih figura kao spriteova.
- import PlayerInput (terminalInputLoop): Uvozi funkciju za rukovanje unosom igrača s terminala.

- `import ChessPieces`: Uvozi definicije vezane uz šahovske figure.
3. **Main Function:**
- `main :: IO ()`: Definira tip glavne funkcije kao IO akciju.
 - `boardRef <- newIORef initialChessboard`: Inicijalizira promjenjivu referencu (`IORef`) za početno stanje šahovske ploče.
 - `colorRef <- newIORef White`: Inicijalizira promjenjivu referencu (`IORef`) za početno stanje boje igrača (Bijela).
 - `forkIO $ terminalInputLoop boardRef colorRef`: Pokreće funkciju za unos s terminala paralelno koristeći `forkIO`.
 - `runProgram boardRef (InWindow "Chessboard" (500, 500) (100, 100))`: Pokreće Gloss simulacijski program s referencom na ploču i postavkama prikaza.
4. **runProgram Function:**
- `runProgram :: IORef Chessboard -> Display -> IO ()`: Definira tip funkcije.
 - `simulateIO display black 10 initialChessboard env2Pic (step boardRef)`: Pokreće Gloss simulaciju:
 - `display`: Postavke prikaza.
 - `black`: Boja pozadine.
 - `10`: Broj simulacijskih koraka u sekundi.
 - `initialChessboard`: Početno stanje šahovske ploče.
 - `env2Pic`: Funkcija za pretvaranje stanja ploče u sliku.
 - `(step boardRef)`: Funkcija za ažuriranje stanja ploče.
5. **env2Pic Function:**
- `env2Pic :: Chessboard -> IO Picture`: Definira tip funkcije.
 - `sprites <- drawChessboardSprites board`: Crta spriteove šahovske ploče.
 - `let boardPic = drawChessboard`: Crta mrežu šahovske ploče.
 - `return $ pictures [boardPic, sprites]`: Kombinira mrežu i spriteove u jednu sliku koristeći `pictures`.
6. **step Function:**
- `step :: IORef Chessboard -> ViewPort -> Float -> Chessboard -> IO Chessboard`: Definira tip funkcije.
 - `step boardRef _ _ _ = readIORef boardRef`: Čita i vraća trenutno stanje šahovske ploče iz `IORef`. Dodatni parametri (`ViewPort`, `Float`, `Chessboard`) se ignoriraju u ovoj implementaciji.

Što je IORef? `IORef` je referentni tip koji omogućava promjenjivo stanje unutar IO monade. U čisto funkcionalnom jeziku poput Haskell, varijable su nepromjenjive po defaultu. Međutim, ponekad je potrebno raditi s promjenjivim stanjem, na primjer u aplikacijama koje uključuju korisnički unos ili grafičke interfejs. `IORef` omogućava promjenjivo stanje na način koji je siguran unutar IO monade.

`newIORef` prima početnu vrijednost kao argument i vraća IO akciju koja stvara novi `IORef` s tom početnom vrijednošću.

Primjer

U gornjem kodu za igru šah, `newIORef` se koristi za stvaranje referenci na stanje šahovske ploče i trenutnu boju igrača.

```
main :: IO ()
main = do
  boardRef <- newIORef initialChessboard -- Stvara novi IORef za šahovsku ploču s
  ↪ početnim stanjem
  colorRef <- newIORef White             -- Stvara novi IORef za boju igrača,
  ↪ postavljajući početnu boju na bijelu
  forkIO $ terminalInputLoop boardRef colorRef -- Pokreće funkciju za unos u zasebnoj
  ↪ dretvi
```

```
runProgram boardRef (InWindow "Chessboard" (500, 500) (100, 100)) -- Pokreće grafičku
↳ simulaciju
```

- `boardRef <- newIORef initialChessboard`:
 - Stvara novi `IORef` koji sadrži početno stanje šahovske ploče definirano u `initialChessboard`.
 - `boardRef` je referenca koja omogućava pristup i modifikaciju šahovske ploče u programu.
- `colorRef <- newIORef White`:
 - Stvara novi `IORef` koji sadrži početno stanje boje igrača, koje je `White` (bijela).
 - `colorRef` je referenca koja omogućava pristup i modifikaciju trenutne boje igrača.

Korištenje `newIORef` omogućava upravljanje promjenjivim stanjem u funkcionalnom jeziku poput Haskell na siguran način unutar IO monade. U primjeru šahovske igre, `IORef` omogućava ažuriranje i čitanje stanja šahovske ploče i trenutne boje igrača tijekom izvršavanja programa.

Što je `forkIO`? `forkIO` je funkcija u Haskellovom modulu `Control.Concurrent` koja omogućava paralelno (konkurentno) izvršavanje IO akcija. Omogućava stvaranje novog lakog procesa (eng. lightweight thread) koji se izvršava istovremeno s glavnim programom.

`forkIO` omogućava pokretanje IO akcije u zasebnoj Haskellovoj dretvi (thread), što znači da se ta akcija može izvršavati paralelno s drugim IO akcijama u programu. Haskellove dretve su lake i učinkovite, omogućujući visok stupanj konkurentnosti.

Sintaksa

```
forkIO :: IO () -> IO ThreadId
```

- Prima IO akciju (tipa `IO ()`) kao argument.
- Vraća `ThreadId`, identifikator novostvorene dretve.

Kako `forkIO` radi? Kada se `forkIO` pozove s IO akcijom, ta akcija se pokreće u novoj dretvi. Glavna nit nastavlja s izvršavanjem ostatka programa bez čekanja da nova nit završi. Ovo je korisno za zadatke koji se mogu izvoditi paralelno, poput rukovanja korisničkim unosom ili rada s mrežom.

Primjer

U primjeru šahovske igre, `forkIO` se koristi za pokretanje funkcije `terminalInputLoop` koja rukuje unosom igrača putem terminala u zasebnoj dretvi, dok se grafička simulacija šahovske ploče izvršava u glavnoj dretvi.

```
main :: IO ()
main = do
  boardRef <- newIORef initialChessboard -- Kreiraj novi IORef koji sadrži početnu
↳ šahovsku ploču
  colorRef <- newIORef White -- Kreiraj novi IORef koji sadrži početnu boju igrača
↳ (Bijela)
  forkIO $ terminalInputLoop boardRef colorRef -- Pokreni petlju za unos s terminala
↳ istovremeno koristeći forkIO
  runProgram boardRef (InWindow "Chessboard" (500, 500) (100, 100)) -- Pokreni
↳ simulacijski program s danom referencom ploče i postavkama prikaza
```

- `forkIO $ terminalInputLoop boardRef colorRef`:
 - Poziva `forkIO` s akcijom `terminalInputLoop boardRef colorRef`.
 - `terminalInputLoop boardRef colorRef` je funkcija koja se izvršava u zasebnoj dretvi.
 - Ova funkcija rukuje korisničkim unosom putem terminala i ažurira stanje šahovske ploče (`boardRef`) i trenutnu boju igrača (`colorRef`).

Prednosti korištenja forkIO

- **Paralelizam:** Omogućava izvršavanje više IO akcija paralelno, što može povećati učinkovitost programa.
- **Jednostavnost:** Korištenje `forkIO` je jednostavno i ne zahtijeva složeno upravljanje dretvi.
- **Brza reakcija:** Omogućava brzu reakciju na korisnički unos ili druge vanjske događaje bez blokiranja glavne dretve.

Što je Gloss?

Gloss je Haskellova biblioteka koja olakšava stvaranje 2D grafike, animacija i simulacija. Dizajnirana je za jednostavnost korištenja i omogućava korisnicima da brzo razviju grafičke aplikacije bez potrebe za detaljnim razumijevanjem rada grafičkih sustava.

Komponente Gloss biblioteke

Graphics.Gloss Ovaj modul pruža osnovne funkcije za crtanje i prikaz 2D grafike.

- **Display (InWindow):**
 - **Display:** Tip koji predstavlja različite načine prikaza prozora.
 - **InWindow:** Konstruktor za prikaz aplikacije u prozoru s određenim nazivom, dimenzijama i pozicijom na ekranu.
- **Picture:**
 - Tip koji predstavlja slike koje se mogu crtati na ekran. Slike mogu biti osnovni oblici (pravokutnici, krugovi), tekst ili složene slike sastavljene od drugih slika.
- **pictures:**
 - Funkcija koja prima listu `Picture` objekata i kombinira ih u jednu sliku. Omogućava crtanje više objekata zajedno.

Graphics.Gloss.Interface.IO.Simulate Ovaj modul pruža funkcije za kreiranje interaktivnih simulacija koje mogu reagirati na korisnički unos i mijenjati se tijekom vremena.

- **Viewport:**
 - Tip koji predstavlja trenutno stanje prikaza, uključujući informacije o povećanju i pomicanju prikaza. Koristi se za transformiranje koordinata tijekom crtanja.
- **simulateIO:**
 - Funkcija koja omogućava stvaranje simulacija koje se ažuriraju i ponovno crtaju na temelju vremena i korisničkog unosa. Koristi se za definiranje simulacijskog programa u Glossu.

```
simulateIO :: Display      -- Postavke prikaza (prozor, fullscreen, itd.)
-> Color                -- Boja pozadine
-> Int                  -- Broj simulacijskih koraka u sekundi
-> world                -- Početno stanje svijeta
-> (world -> IO Picture) -- Funkcija za crtanje trenutnog stanja svijeta
-> (Viewport -> Float -> world -> IO world) -- Funkcija za ažuriranje
    ⇨ stanja svijeta
-> IO ()
```

Primjer

U `main.hs` kodu, **Gloss** biblioteka se koristi za stvaranje prozora u kojem će se prikazivati šahovska ploča i figure. `simulateIO` funkcija se koristi za pokretanje simulacije koja kontinuirano osvježava prikaz na temelju trenutnog stanja šahovske ploče.

- **Display (InWindow):**
 - Koristi se za definiranje prozora aplikacije sa specifičnim nazivom, dimenzijama i pozicijom:
`InWindow "Chessboard" (500, 500) (100, 100)`
- **Picture:**

- Koristi se za stvaranje i kombiniranje slika šahovske ploče i figura:

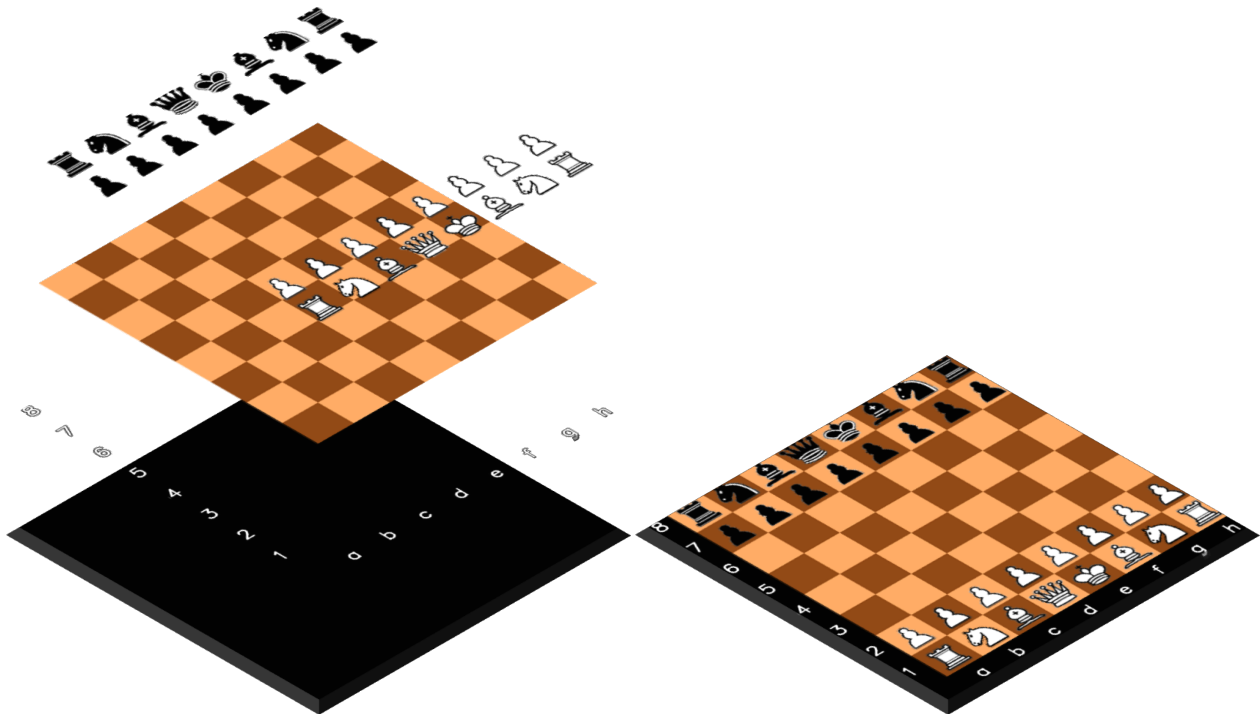

```
let boardPic = drawChessboard
return $ pictures [boardPic, sprites]
```
- **black:**
 - Postavlja boju pozadine prozora na crnu:


```
simulateIO display black 10 initialChessboard env2Pic (step boardRef)
```
- **pictures:**
 - Kombinira mrežu šahovske ploče i figure u jednu sliku:


```
return $ pictures [boardPic, sprites]
```
- **simulateIO:**
 - Pokreće simulaciju koja kontinuirano ažurira i prikazuje šahovsku ploču:


```
simulateIO display black 10 initialChessboard env2Pic (step boardRef)
```

Primjer odvojenih slika i kombinirane slike:



- **Odvojene slika:** Prikazuje pozadinu, labele, šahovsku ploču, figure kao zasebne slika.
- **Kombinirana slika:** Prikazuje kombinirane slika u jednu sliku koristeći Gloss biblioteku.

Korištenjem funkcije `pictures`, pojedinačne slike (kao što su labele, ploča i figure) mogu se kombinirati u jednu sliku koja se zatim prikazuje korisniku. Ovo omogućava dinamičko i interaktivno ažuriranje prikaza na temelju poteza igrača.

ChessPieces

Modul `ChessPieces.hs` u projektu za šahovsku igru definira osnovne tipove podataka i funkcije povezane sa šahovskim figurama i njihovim karakteristikama. Konkretno, modul pruža:

1. **Definiciju Boja:**

- Modul definira tip podataka `Color`, koji predstavlja boje šahovskih figura. Dostupne boje su `White` (bijela) i `Black` (crna).

2. **Definiciju Šahovskih Figura:**

- Modul definira tip podataka `Piece`, koji predstavlja različite vrste šahovskih figura (kralj, kraljica, top, lovac, konj i pješak). Svaka figura je povezana s određenom bojom (`White` ili `Black`).

3. **Definiciju Polja na Šahovskoj Ploči:**

- Modul definira tip podataka `Square`, koji predstavlja polje na šahovskoj ploči. Polje može biti `Empty` (prazno) ili `Occupied` (zauzeto figurom). Ako je polje zauzeto, čuva se informacija o figuri koja zauzima to polje.

4. **Funkciju za Dobivanje Boje Figure:**

- Modul pruža funkciju `pieceColor`, koja prima šahovsku figuru i vraća boju te figure. Ovo je korisno za određivanje kojoj boji pripada određena figura tijekom igre.

```
module ChessPieces (Color(..), Piece(..), Square(..), pieceColor) where

-- Modul deklarira koje tipove i funkcije izvozi: Color, Piece, Square i pieceColor

-- Tip podataka koji predstavlja boje šahovskih figura
data Color = White | Black deriving (Eq, Show)
-- data Color definira tip podataka s dvije moguće vrijednosti: White i Black
-- deriving (Eq, Show) automatski generira instance Eq i Show tip klasa za usporedbu i
--   ispis boja

-- Tip podataka koji predstavlja šahovske figure
data Piece = King Color | Queen Color | Rook Color | Bishop Color | Knight Color | Pawn
           -- Color deriving (Eq, Show)
-- data Piece definira tip podataka s konstruktorima za svaku figuru (King, Queen, Rook,
--   Bishop, Knight, Pawn)
-- Svaki konstruktor prima argument tipa Color, čime se specificira boja figure
-- deriving (Eq, Show) automatski generira instance Eq i Show tip klasa za usporedbu i
--   ispis figura

-- Tip podataka koji predstavlja polje na šahovskoj ploči
data Square = Empty | Occupied Piece deriving (Eq, Show)
-- data Square definira tip podataka s dvije moguće vrijednosti: Empty i Occupied
-- Konstruktor Occupied prima argument tipa Piece, čime se specificira koja figura
--   zauzima to polje
-- deriving (Eq, Show) automatski generira instance Eq i Show tip klasa za usporedbu i
--   ispis polja

-- Funkcija koja vraća boju figure
pieceColor :: Piece -> Color
pieceColor (King color)    = color
-- Za figuru King, funkcija vraća boju koja je pridružena kralju
pieceColor (Queen color)   = color
-- Za figuru Queen, funkcija vraća boju koja je pridružena kraljici
pieceColor (Rook color)     = color
-- Za figuru Rook, funkcija vraća boju koja je pridružena topu
pieceColor (Bishop color)  = color
```

```
-- Za figuru Bishop, funkcija vraća boju koja je pridružena lovcu
pieceColor (Knight color) = color
-- Za figuru Knight, funkcija vraća boju koja je pridružena konju
pieceColor (Pawn color)   = color
-- Za figuru Pawn, funkcija vraća boju koja je pridružena pješaku
```

1. Deklaracije Modula:

- module ChessPieces (Color(..), Piece(..), Square(..), pieceColor) where:
 - Ova linija definira modul ChessPieces i specificira koje tipove podataka i funkcije modul izvozi: Color, Piece, Square i pieceColor.

2. Tip Podatka Boje:

- data Color = White | Black deriving (Eq, Show):
 - Definira tip podataka Color koji može biti White ili Black.
 - deriving (Eq, Show) automatski generira instance tipova Eq (za usporedbu vrijednosti) i Show (za pretvaranje u niz za ispis).

3. Tip Podatka Figure:

- data Piece = King Color | Queen Color | Rook Color | Bishop Color | Knight Color | Pawn Color deriving (Eq, Show):
 - Definira tip podataka Piece s konstruktorima za svaku vrstu šahovske figure (King, Queen, Rook, Bishop, Knight, Pawn), svaki s argumentom tipa Color.
 - deriving (Eq, Show) automatski generira instance tipova Eq i Show za Piece.

4. Tip Podatka Polja:

- data Square = Empty | Occupied Piece deriving (Eq, Show):
 - Definira tip podataka Square koji može biti Empty (prazno) ili Occupied (zauzeto figurom tipa Piece).
 - deriving (Eq, Show) automatski generira instance tipova Eq i Show za Square.

5. Dohvaćanje Boje Figure:

- pieceColor :: Piece -> Color:
 - Tip funkcije pieceColor koja prima argument tipa Piece i vraća vrijednost tipa Color.
- pieceColor (King color) = color i slične linije za ostale figure:
 - Funkcija koristi obrasce za raspakiravanje konstruktora Piece i vraća pridruženu boju (color).
 - Za svaku vrstu figure (King, Queen, Rook, Bishop, Knight, Pawn), funkcija vraća boju koja je pridružena toj figuri.

Namjena Modula

Modul ChessPieces.hs služi kao temelj za definiranje osnovnih elemenata igre šaha. Tipovi podataka i funkcije definirani u ovom modulu koriste se u drugim dijelovima programa za rad s figurama i poljima na ploči. Na primjer, informacije o figurama i njihovim bojama koriste se za provjeru valjanosti poteza, crtanje ploče i figura te logiku igre.

ChessSprites



Modul `ChessSprites.hs` u projektu za šahovsku igru odgovoran je za grafički prikaz šahovskih figura koristeći `Gloss` biblioteku. Konkretno, modul pruža funkcionalnosti za:

1. Učitavanje Slika Figura:

- Modul učitava slike (spriteove) različitih šahovskih figura iz datoteka. Svaka figura (bijela i crna) ima svoju odgovarajuću sliku koja se koristi za prikaz na ploči.

2. Crtanje Figura na Šahovskoj Ploči:

- Modul definira funkcije za crtanje šahovskih figura na njihovim odgovarajućim pozicijama na ploči. Koristeći `Gloss` funkcije za transformaciju i prikaz slika, figure se prikazuju na točnim koordinatama ploče.

3. Kombiniranje Mreže i Figura:

- Modul kombinira mrežu šahovske ploče i spriteove figura u jednu sliku koja se prikazuje korisniku. Ovo omogućava da se šahovska ploča i figure prikazuju zajedno u jednom prozoru aplikacije.

```
module ChessSprites (drawChessboardSprites) where

-- Uvoz potrebnih modula
import Graphics.Gloss -- Uvoz Gloss biblioteke za grafiku
import ChessPieces    -- Uvoz definicija šahovskih figura i boja
import Chessboard    -- Uvoz definicija šahovske ploče

-- Funkcija za učitavanje slika šahovskih figura
loadImages :: IO [(Piece, Picture)]
loadImages = do
    -- Definiranje lista figura i imena datoteka za bijele figure
    let whitePieces = [ (King White, "white_king")
                      , (Queen White, "white_queen")
                      , (Rook White, "white_rook")
                      , (Bishop White, "white_bishop")
                      , (Knight White, "white_knight")
                      , (Pawn White, "white_pawn") ]
    -- Definiranje lista figura i imena datoteka za crne figure
    blackPieces = [ (King Black, "black_king")
                  , (Queen Black, "black_queen")
                  , (Rook Black, "black_rook")
                  , (Bishop Black, "black_bishop")
                  , (Knight Black, "black_knight")
                  , (Pawn Black, "black_pawn") ]
    -- Učitavanje slika za bijele figure i stvaranje parova (figura, slika)
    whiteImages <- mapM (\(piece, fileName) -> (piece,) <$> loadBMP ("sprites/" ++
    ↪ fileName ++ ".bmp")) whitePieces
    -- Učitavanje slika za crne figure i stvaranje parova (figura, slika)
    blackImages <- mapM (\(piece, fileName) -> (piece,) <$> loadBMP ("sprites/" ++
    ↪ fileName ++ ".bmp")) blackPieces
    -- Vraćanje kombinirane liste bijelih i crnih slika
    return $ whiteImages ++ blackImages

-- Funkcija za crtanje šahovske ploče sa spriteovima/slikama
```

```

drawChessboardSprites :: Chessboard -> IO Picture
drawChessboardSprites board = do
    -- Učitavanje slika figura
    images <- loadImages
    -- Veličina kvadrata na ploči
    let squareSize = 50 -- Podešavanje po potrebi
        -- Pomicanje ploče za centriranje
        xOffset = fromIntegral $ negate $ squareSize * 4 - 25
        yOffset = fromIntegral $ negate $ squareSize * 4 - 25
        -- Stvaranje liste slika za svako polje na ploči
        spriteBoard = [ [ case sq of
            Empty -> Blank -- Ako je polje prazno, nema slike
            -- Ako je polje zauzeto, postavi sliku figure na odgovarajuću
            -- poziciju
            Occupied piece -> translate (fromIntegral (x * squareSize) +
            xOffset)
                                (fromIntegral (y * squareSize) +
            yOffset)
            | (x, sq) <- zip [0..] row ] -- Iteracija kroz redove i stupce
            | (y, row) <- zip [0..] board ] -- Iteracija kroz redove
        -- Vraćanje kombinirane slike ploče i figura
    return $ pictures $ concat spriteBoard

```

1. Deklaracije Modula:

- module ChessSprites (drawChessboardSprites) where: Definira modul ChessSprites i izvozi funkciju drawChessboardSprites.

2. Uvozi:

- import Graphics.Gloss: Uvoz Gloss biblioteke za rad s grafikom.
- import ChessPieces: Uvoz modula za rad s šahovskim figurama i bojama.
- import Chessboard: Uvoz modula za rad sa šahovskom pločom.

3. Učitavanje Slika:

- loadImages :: IO [(Piece, Picture)]: Definira funkciju koja vraća IO akciju koja učitava slike figura i vraća listu parova (figura, slika).
- let whitePieces = ...: Definira listu bijelih figura i pripadajućih imena datoteka.
- let blackPieces = ...: Definira listu crnih figura i pripadajućih imena datoteka.
- whiteImages <- mapM ...: Učitava slike za bijele figure i stvara parove (figura, slika) koristeći loadBMP.
- blackImages <- mapM ...: Učitava slike za crne figure i stvara parove (figura, slika) koristeći loadBMP.
- return \$ whiteImages ++ blackImages: Vraća kombiniranu listu bijelih i crnih slika figura.

4. Crtanje Figura:

- drawChessboardSprites :: Chessboard -> IO Picture: Definira funkciju koja prima šahovsku ploču i vraća IO akciju koja stvara sliku ploče sa figurama.
- images <- loadImages: Učitava slike figura.
- let squareSize = 50: Definira veličinu kvadrata na ploči.
- let xOffset = ...: Računa horizontalni pomak za centriranje ploče.
- let yOffset = ...: Računa vertikalni pomak za centriranje ploče.
- let spriteBoard = ...: Stvara listu slika za svako polje na ploči:
 - case sq of Empty -> Blank: Ako je polje prazno, nema slike.
 - Occupied piece -> translate ...: Ako je polje zauzeto, postavi sliku figure na odgovarajuću poziciju koristeći translate.
 - | (x, sq) <- zip [0..] row: Iteracija kroz stupce unutar reda.

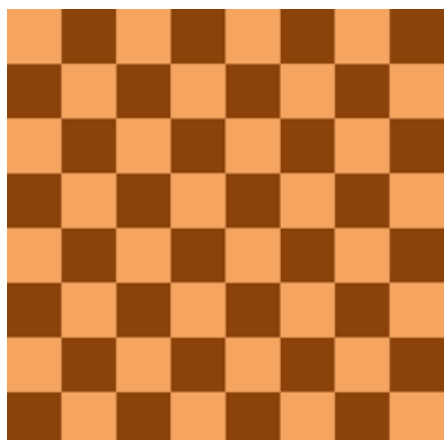
- | (y, row) <- zip [0..] board: Iteracija kroz redove ploče.
- return \$ pictures \$ concat spriteBoard: Vraća kombiniranu sliku ploče i figura koristeći pictures i concat za spajanje svih slika u jednu.



Namjena Modula

Modul `ChessSprites.hs` služi za stvaranje vizualnog prikaza šahovske igre. Njegove funkcionalnosti omogućavaju da se šahovske figure pravilno prikazuju na ploči i ažuriraju prema potezima igrača. Korištenjem ovog modula, grafički prikaz igre postaje dinamičan i interaktivan, omogućavajući igračima da jasno vide stanje igre u svakom trenutku.

Chessboard



Modul `Chessboard.hs` u projektu za šahovsku igru odgovoran je za definiciju i manipulaciju šahovske ploče. Konkretno, modul pruža funkcionalnosti za:

1. **Definiciju Šahovske Ploče:**

- Definira tip podataka `Chessboard`, koji predstavlja šahovsku ploču kao dvodimenzionalni popis (`list`) polja (`Square`).

2. **Inicijalizacija Početnog Stanja Ploče:**

- Pruža funkciju `initialChessboard`, koja vraća početno stanje šahovske ploče s postavljenim figurama na početne pozicije.

3. **Crtanje Šahovske Ploče:**

- Definira funkciju `drawChessboard`, koja koristi `Gloss` biblioteku za crtanje mreže šahovske ploče.

4. **Dohvaćanje i Provjera Stanja Polja:**

- Pruža funkcije za dohvaćanje stanja specifičnih polja na ploči (`pieceAt`) i provjeru da li je polje prazno (`isEmpty`).

```
module Chessboard (Chessboard, initialChessboard, drawChessboard, pieceAt, isEmpty) where
```

```
-- Uvoz potrebnih modula
```

```
import Graphics.Gloss -- Uvoz Gloss biblioteke za rad s grafikom
```

```
import ChessPieces -- Uvoz modula ChessPieces za rad sa šahovskim figurama i poljima
```

```
-- Tip sinonim za red šahovske ploče
```

```
type Row = [Square]
```

```
-- Tip sinonim za šahovsku ploču
```

```
type Chessboard = [Row]
```

```
-- Početno stanje šahovske ploče s figurama postavljenim na početne pozicije
```

```
initialChessboard :: Chessboard
```

```
initialChessboard =
```

```
  [ [ Occupied (Rook White), Occupied (Knight White), Occupied (Bishop White), Occupied
    ↪ (Queen White)
    , Occupied (King White), Occupied (Bishop White), Occupied (Knight White), Occupied
    ↪ (Rook White) ]
    , replicate 8 (Occupied (Pawn White)) -- Drugi red je popunjen bijelim pješacima
    , replicate 8 Empty -- Treći do šesti redovi su prazni
    , replicate 8 Empty
    , replicate 8 Empty
    , replicate 8 Empty
```



```

    , replicate 8 (Occupied (Pawn Black)) -- Sedmi red je popunjen crnim pješacima
    , [ Occupied (Rook Black), Occupied (Knight Black), Occupied (Bishop Black), Occupied
    ↪ (Queen Black)
    , Occupied (King Black), Occupied (Bishop Black), Occupied (Knight Black), Occupied
    ↪ (Rook Black) ]
  ]

-- Funkcija za crtanje šahovske ploče
drawChessboard :: Picture
drawChessboard = pictures (
  [drawSquare x y | x <- [0..7], y <- [0..7]] ++ -- Crtanje kvadrata za svaki red i
  ↪ stupac
  [drawLabel x y | x <- [0..7], y <- [0..7]] ++ -- Crtanje oznaka za stupce (a-h)
  [drawSideLabel x | x <- [0..7]] -- Crtanje oznaka za redove (1-8)
)

-- Funkcija za crtanje pojedinog kvadrata šahovske ploče
drawSquare :: Int -> Int -> Picture
drawSquare x y = translate (fromIntegral x * 50 - 175) (fromIntegral y * 50 - 175) $
  color (if even (x + y) then darkBrown else lightBrown) $ --
  ↪ Naizmjenično bojanje kvadrata
  rectangleSolid 50 50 -- Crtanje kvadrata veličine 50x50

-- Definiranje prilagođenih boja
darkBrown :: Graphics.Gloss.Color
darkBrown = makeColorI 139 69 19 255 -- Smeđa boja (Saddle Brown)

lightBrown :: Graphics.Gloss.Color
lightBrown = makeColorI 244 164 96 255 -- Svijetlosmeđa boja (Sandy Brown)

-- Funkcija za crtanje oznaka za stupce (a-h)
drawLabel :: Int -> Int -> Picture
drawLabel x _ = translate (fromIntegral x * 50 - 175) (-225) $ -- Pomicanje oznake na
  ↪ odgovarajuću poziciju
  scale 0.15 0.15 $ color white $ text [toEnum (fromEnum 'a' + x)] --
  ↪ Crtanje oznake stupca

-- Funkcija za crtanje oznaka za redove (1-8)
drawSideLabel :: Int -> Picture
drawSideLabel y = translate (-225) (fromIntegral y * 50 - 175) $ -- Pomicanje oznake na
  ↪ odgovarajuću poziciju
  scale 0.15 0.15 $ color white $ text (show (9 - (8 - y))) -- Crtanje
  ↪ oznake reda

-- Funkcija za dohvaćanje figure na danoj poziciji
pieceAt :: Chessboard -> (Int, Int) -> Maybe Piece
pieceAt board (x, y) = case board !! y !! x of -- Dohvaćanje polja na danoj poziciji
  Occupied piece -> Just piece -- Ako je polje zauzeto, vraća se figura
  _ -> Nothing -- Ako je polje prazno, vraća se Nothing

-- Pomoćna funkcija za provjeru je li polje prazno
isEmpty :: Square -> Bool
isEmpty Empty = True -- Ako je polje prazno, vraća True
isEmpty _ = False -- U suprotnom, vraća False

```

1. Deklaracije Modula:

- `module Chessboard (Chessboard, initialChessboard, drawChessboard, pieceAt, isEmpty) where:`
 - Definira modul `Chessboard` i specificira koje tipove i funkcije izvozi: `Chessboard`, `initialChessboard`, `drawChessboard`, `pieceAt`, `isEmpty`.

2. Uvozi:

- `import Graphics.Gloss:` Uvoz `Gloss` biblioteke za rad s grafikom.
- `import ChessPieces:` Uvoz modula `ChessPieces` za rad sa šahovskim figurama i poljima.

3. Sinonimi Tipova:

- `type Row = [Square]:` Definira `Row` kao sinonim za listu `Square`.
- `type Chessboard = [Row]:` Definira `Chessboard` kao sinonim za listu redova (`Row`).

4. inicijalna Ploča:

- `initialChessboard :: Chessboard:` Definira početno stanje šahovske ploče.
- Popis polja na šahovskoj ploči s figurama postavljenim na početne pozicije:
 - Bijele figure u prvom redu.
 - Bijeli pješaci u drugom redu.
 - Prazna polja u sredini.
 - Crni pješaci u sedmom redu.
 - Crne figure u osmom redu.

5. Crtanje Ploče:

- `drawChessboard :: Picture:` Funkcija koja crta šahovsku ploču koristeći `Gloss`.
- Kombinira slike kvadrata, oznaka stupaca i oznaka redova u jednu sliku pomoću `pictures`.

6. Crtanje Polja:

- `drawSquare :: Int -> Int -> Picture:` Funkcija koja crta pojedini kvadrat šahovske ploče.
- Koristi `translate` za postavljanje kvadrata na odgovarajuću poziciju.
- Koristi `color` i `rectangleSolid` za crtanje kvadrata odgovarajuće boje.

7. Boje Polja:

- `darkBrown` i `lightBrown:` Definiraju prilagođene boje za kvadrate šahovske ploče.



8. Crtanje Labela:

- `drawLabel :: Int -> Int -> Picture:` Funkcija koja crta oznake stupaca (a-h).
- `drawSideLabel :: Int -> Picture:` Funkcija koja crta oznake redova (1-8).

8

7

6

5

4

3

2

1

a b c d e f g h

9. Dohvaćanje Figure na Poziciji:

- `pieceAt :: Chessboard -> (Int, Int) -> Maybe Piece`: Funkcija koja dohvaća figuru na danoj poziciji na ploči.
 - Koristi indeksiranje da bi dohvatila polje i vraća figuru ako je polje zauzeto.
10. **Provjera Praznog Polje:**
- `isEmpty :: Square -> Bool`: Pomoćna funkcija koja provjerava je li polje prazno (`Empty`).

Namjena Modula

Modul `Chessboard.hs` služi kao temelj za sve operacije vezane uz šahovsku ploču. Njegove funkcionalnosti omogućavaju:

- Postavljanje početnog stanja igre.
- Crtanje ploče na grafičkom sučelju.
- Manipulaciju pločom tijekom igre, uključujući provjeru i promjenu stanja pojedinih polja.

Ovaj modul je ključan za logiku igre, jer omogućava centralizirano upravljanje šahovskom pločom i interakcijama figura na njoj.

ChessLogic

Modul `ChessLogic.hs` u projektu za šahovsku igru odgovoran je za implementaciju logike igre. Konkretno, modul pruža funkcionalnosti za:

1. Validaciju Poteza:

- Provjerava jesu li potezi figura valjani prema pravilima šaha. Ovo uključuje provjeru specifičnih pravila za svaku figuru, kao što su dozvoljeni pokreti za kralja, kraljicu, topa, lovca, konja i pješaka.

2. Izvođenje Poteza:

- Ažurira stanje šahovske ploče nakon što je potez validiran kao ispravan. Ovo uključuje pomicanje figura i uklanjanje figura koje su pojedene.

3. Provjeru i Upravljanje Stanjem Igrača:

- Upravlja izmjenom poteza između bijelog i crnog igrača. Osigurava da se igrači izmjenjuju pravilno i prati koja je boja trenutno na potezu.

4. Provjeru Specifičnih Stanja:

- Provjerava specifične uvjete igre kao što su šah i mat, te druge specifične poteze poput en passant hvatanja i promocije pješaka.

```
module ChessLogic (makeMove, isValidMove, switchColor) where

-- Uvoz potrebnih modula
import Chessboard (Chessboard, pieceAt) -- Uvoz modula Chessboard za rad sa šahovskom
    ⇨ pločom
import ChessPieces -- Uvoz modula ChessPieces za rad sa šahovskim figurama

-- Funkcija za promjenu trenutnog igrača
switchColor :: Color -> Color
switchColor White = Black -- Ako je trenutna boja bijela, promijeni na crnu
switchColor Black = White -- Ako je trenutna boja crna, promijeni na bijelu

-- Funkcija za obavljanje poteza na šahovskoj ploči ako je potez valjan
makeMove :: Color -> (Int, Int) -> (Int, Int) -> Chessboard -> Maybe Chessboard
makeMove color (fromX, fromY) (toX, toY) board
    -- Provjerava je li potez unutar granica ploče, valjan i je li figura na početnoj
    ⇨ poziciji odgovarajuće boje
    | inBounds (fromX, fromY) && inBounds (toX, toY) &&
      isValidMove board (fromX, fromY) (toX, toY) && pieceColorAt board (fromX, fromY) ==
    ⇨ Just color =
        let piece = board !! fromY !! fromX -- Dohvati figuru na početnoj poziciji
            -- Ažuriraj redak na ciljnoj poziciji s figurom
            updatedRow row idx newSquare = take idx row ++ [newSquare] ++ drop (idx + 1)
        ⇨ row
            -- Ažuriraj ploču s pomaknutom figurom na ciljnu poziciju
            updatedBoard = take toY board ++
                [updatedRow (board !! toY) toX piece] ++
                drop (toY + 1) board
            -- Očisti početnu poziciju figure
            clearedBoard = take fromY updatedBoard ++
                [updatedRow (updatedBoard !! fromY) fromX Empty] ++
                drop (fromY + 1) updatedBoard
            in Just clearedBoard -- Vрати ažuriranu ploču
    | otherwise = Nothing -- Ako je potez nevaljan, vrati Nothing

-- Provjerava je li pozicija unutar granica šahovske ploče
```

```

inBounds :: (Int, Int) -> Bool
inBounds (x, y) = x >= 0 && x < 8 && y >= 0 && y < 8 -- Ploča je 8x8, pa x i y moraju
    ↳ biti u rasponu 0-7

-- Funkcija za dohvaćanje boje figure na danoj poziciji
pieceColorAt :: Chessboard -> (Int, Int) -> Maybe Color
pieceColorAt board (x, y) = case board !! y !! x of
    Occupied piece -> Just (pieceColor piece) -- Ako je polje zauzeto, vrati boju figure
    _ -> Nothing -- Ako je polje prazno, vrati Nothing

-- Funkcija za validaciju poteza određene figure
isValidMove :: Chessboard -> (Int, Int) -> (Int, Int) -> Bool
isValidMove board (x1, y1) (x2, y2) = case pieceAt board (x1, y1) of
    Just piece ->
        let color = pieceColor piece -- Dohvati boju figure
        in case pieceAt board (x2, y2) of
            Just destPiece -> pieceColor destPiece /= color && isValidPieceMove piece (x1,
                ↳ y1) (x2, y2) board -- Provjeri je li ciljna pozicija zauzeta protivničkom
                ↳ figurom i je li potez valjan
            Nothing -> isValidPieceMove piece (x1, y1) (x2, y2) board -- Ako je ciljna
                ↳ pozicija prazna, provjeri je li potez valjan
    Nothing -> False -- Ako na početnoj poziciji nema figure, potez nije valjan

-- Funkcija za provjeru valjanosti poteza specifične figure
isValidPieceMove :: Piece -> (Int, Int) -> (Int, Int) -> Chessboard -> Bool
isValidPieceMove (Pawn color) = isValidPawnMove color -- Provjeri valjanost poteza
    ↳ pješaka
isValidPieceMove (Rook color) = isValidRookMove color -- Provjeri valjanost poteza topa
isValidPieceMove (Knight color) = isValidKnightMove color -- Provjeri valjanost poteza
    ↳ konja
isValidPieceMove (Bishop color) = isValidBishopMove color -- Provjeri valjanost poteza
    ↳ lovca
isValidPieceMove (Queen color) = isValidQueenMove color -- Provjeri valjanost poteza
    ↳ kraljice
isValidPieceMove (King color) = isValidKingMove color -- Provjeri valjanost poteza
    ↳ kralja

-- Validacija poteza pješaka (pojednostavljena verzija)
isValidPawnMove :: Color -> (Int, Int) -> (Int, Int) -> Chessboard -> Bool
isValidPawnMove color (x1, y1) (x2, y2) board =
    let direction = if color == White then 1 else -1 -- Smjer kretanja pješaka ovisno o
        ↳ boji
        startRow = if color == White then 1 else 6 -- Početni red pješaka ovisno o boji
        isForwardMove = x1 == x2 && (y2 - y1 == direction || (y1 == startRow && y2 - y1 ==
            ↳ 2 * direction)) -- Provjera je li potez naprijed
        isCaptureMove = abs (x2 - x1) == 1 && y2 - y1 == direction && isOccupiedByOpponent
            ↳ color board (x2, y2) -- Provjera je li potez hvatanje protivničke figure
        in isForwardMove || isCaptureMove -- Potez je valjan ako je naprijed ili hvatanje

-- Validacija poteza topa
isValidRookMove :: Color -> (Int, Int) -> (Int, Int) -> Chessboard -> Bool
isValidRookMove color (x1, y1) (x2, y2) board =
    (x1 == x2 || y1 == y2) && pathIsClear board (x1, y1) (x2, y2) -- Potez je valjan ako
    ↳ je u istom stupcu ili redu i putanja je čista

```

```

-- Validacija poteza konja
isValidKnightMove :: Color -> (Int, Int) -> (Int, Int) -> Chessboard -> Bool
isValidKnightMove color (x1, y1) (x2, y2) _ =
    (abs (x2 - x1) == 2 && abs (y2 - y1) == 1) || (abs (x2 - x1) == 1 && abs (y2 - y1) ==
    ↪ 2) -- Potez je valjan ako je u obliku slova "L"

-- Validacija poteza lovca
isValidBishopMove :: Color -> (Int, Int) -> (Int, Int) -> Chessboard -> Bool
isValidBishopMove color (x1, y1) (x2, y2) board =
    abs (x2 - x1) == abs (y2 - y1) && pathIsClear board (x1, y1) (x2, y2) -- Potez je
    ↪ valjan ako se kreće dijagonalno i putanja je čista

-- Validacija poteza kraljice
isValidQueenMove :: Color -> (Int, Int) -> (Int, Int) -> Chessboard -> Bool
isValidQueenMove color (x1, y1) (x2, y2) board =
    isValidRookMove color (x1, y1) (x2, y2) board || isValidBishopMove color (x1, y1) (x2,
    ↪ y2) board -- Kraljica se može kretati kao top ili lovac

-- Validacija poteza kralja
isValidKingMove :: Color -> (Int, Int) -> (Int, Int) -> Chessboard -> Bool
isValidKingMove color (x1, y1) (x2, y2) _ =
    abs (x2 - x1) <= 1 && abs (y2 - y1) <= 1 -- Kralj se može kretati za jedno polje u
    ↪ bilo kojem smjeru

-- Pomoćne funkcije
-- Proučavanje je li polje zauzeto protivničkom figurom
isOccupiedByOpponent :: Color -> Chessboard -> (Int, Int) -> Bool
isOccupiedByOpponent color board (x, y) = case board !! y !! x of
    Occupied piece -> pieceColor piece /= color -- Ako je polje zauzeto figurom
    ↪ protivnika, vraća True
    _ -> False -- Inače vraća False

-- Proučavanje je li putanja kretanja čista (nema drugih figura na putu)
pathIsClear :: Chessboard -> (Int, Int) -> (Int, Int) -> Bool
pathIsClear board (x1, y1) (x2, y2) =
    let deltaX = signum (x2 - x1) -- Smjer kretanja po x osi
        deltaY = signum (y2 - y1) -- Smjer kretanja po y osi
        path = takeWhile (/= (x2, y2)) $ tail $ zip (iterate (+ deltaX) x1) (iterate (+
    ↪ deltaY) y1) -- Generiranje putanje između početne i ciljne pozicije
    in all (\(x, y) -> board !! y !! x == Empty) path -- Proučava je li svako polje na
    ↪ putu prazno

```

1. Deklaracije Modula:

- module ChessLogic (makeMove, isValidMove, switchColor) where: Definira modul ChessLogic i izvozi funkcije makeMove, isValidMove i switchColor.

2. Uvozi:

- import Chessboard (Chessboard, pieceAt): Uvoz modula Chessboard za rad sa šahovskom pločom.
- import ChessPieces: Uvoz modula ChessPieces za rad sa šahovskim figurama.

3. Promjena Boje Igrača:

- switchColor :: Color -> Color: Funkcija koja mijenja trenutnog igrača.
- switchColor White = Black: Ako je trenutna boja bijela, promijeni na crnu.
- switchColor Black = White: Ako je trenutna boja crna, promijeni na bijelu.

4. Potez:

- `makeMove :: Color -> (Int, Int) -> (Int, Int) -> Chessboard -> Maybe Chessboard`: Funkcija koja obavlja potez na šahovskoj ploči ako je potez valjan.
- `inBounds (fromX, fromY) && inBounds (toX, toY)`: Provjera je li potez unutar granica ploče.
- `isValidMove board (fromX, fromY) (toX, toY)`: Provjera je li potez valjan.
- `pieceColorAt board (fromX, fromY) == Just color`: Provjera je li figura na početnoj poziciji odgovarajuće boje.
- `updatedRow row idx newSquare`: Ažuriranje retka na ciljnoj poziciji s figurom.
- `updatedBoard`: Ažuriranje ploče s pomaknutom figurom na ciljnu poziciju.
- `clearedBoard`: Očisti početnu poziciju figure.
- `Just clearedBoard`: Vraća ažuriranu ploču ako je potez valjan.
- `Nothing`: Ako je potez nevaljan, vrati `Nothing`.

5. Provjera Pozicije:

- `inBounds :: (Int, Int) -> Bool`: Funkcija koja provjerava je li pozicija unutar granica šahovske ploče.
- `x >= 0 && x < 8 && y >= 0 && y < 8`: Ploča je 8x8, pa x i y moraju biti u rasponu 0-7.

6. Boja Figure Na Poziciji:

- `pieceColorAt :: Chessboard -> (Int, Int) -> Maybe Color`: Funkcija koja dohvaća boju figure na danoj poziciji.
- `Occupied piece -> Just (pieceColor piece)`: Ako je polje zauzeto, vrati boju figure.
- `_ -> Nothing`: Ako je polje prazno, vrati `Nothing`.

7. Validacija Poteza:

- `isValidMove :: Chessboard -> (Int, Int) -> (Int, Int) -> Bool`: Funkcija koja validira potez određene figure.
- `pieceAt board (x1, y1)`: Dohvaćanje figure na početnoj poziciji.
- `pieceColor piece`: Dohvaćanje boje figure.
- `pieceColor destPiece /= color`: Provjera je li ciljna pozicija zauzeta protivničkom figurom.
- `isValidPieceMove piece (x1, y1) (x2, y2) board`: Provjera je li potez valjan za specifičnu figuru.

8. Validacija Micanja Figure:

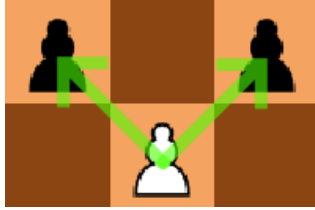
- `isValidPieceMove :: Piece -> (Int, Int) -> (Int, Int) -> Chessboard -> Bool`: Funkcija koja provjerava valjanost poteza specifične figure.
- `isValidPawnMove color`: Provjera valjanosti poteza pješaka.
- `isValidRookMove color`: Provjera valjanosti poteza topa.
- `isValidKnightMove color`: Provjera valjanosti poteza konja.
- `isValidBishopMove color`: Provjera valjanosti poteza lovca.
- `isValidQueenMove color`: Provjera valjanosti poteza kraljice.
- `isValidKingMove color`: Provjera valjanosti poteza kralja.

9. Validacija poteza specifičnih figura:

- `isValidPawnMove`:
 - Provjerava valjanost poteza pješaka, uzimajući u obzir smjer kretanja (naprijed za bijele, nazad za crne), početni red i mogućnost hvatanja figura protivnika dijagonalno.
 - Ako je pješak u početnom redu, može se pomaknuti za dva polja unaprijed, pod uvjetom da su oba polja prazna

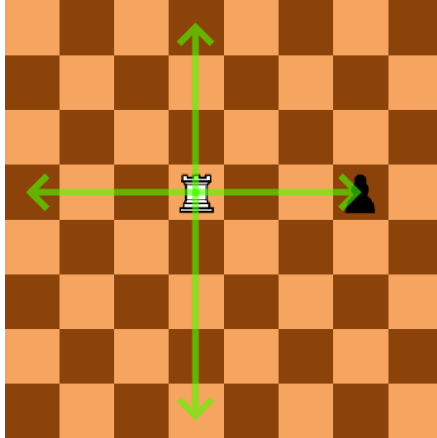


- Hvatanje protivničke figure moguće je samo dijagonalnim potezom u jednom polju unaprijed.



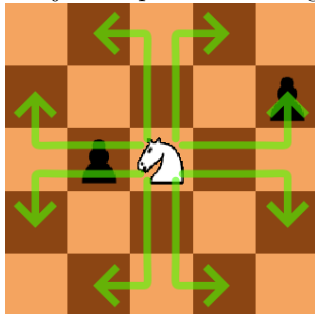
- **isValidRookMove:**

- Provjerava valjanost poteza topa, koji se može kretati horizontalno ili vertikalno bilo kojim brojem polja, pod uvjetom da su sva polja na putu prazna.
- Ako je putanja slobodna, potez je valjan.



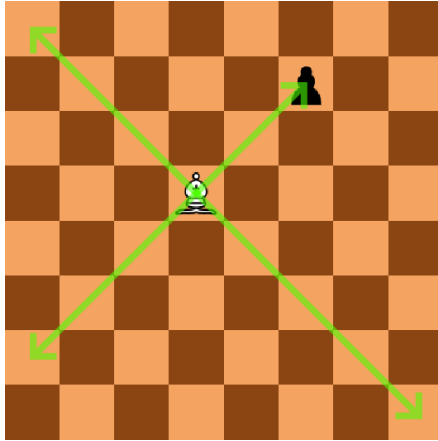
- **isValidKnightMove:**

- Provjerava valjanost poteza konja, koji se može kretati u obliku slova “L” (dva polja u jednom smjeru, zatim jedno polje okomito, ili jedno polje u jednom smjeru, zatim dva polja okomito).
- Konj može preskakati druge figure na ploči.



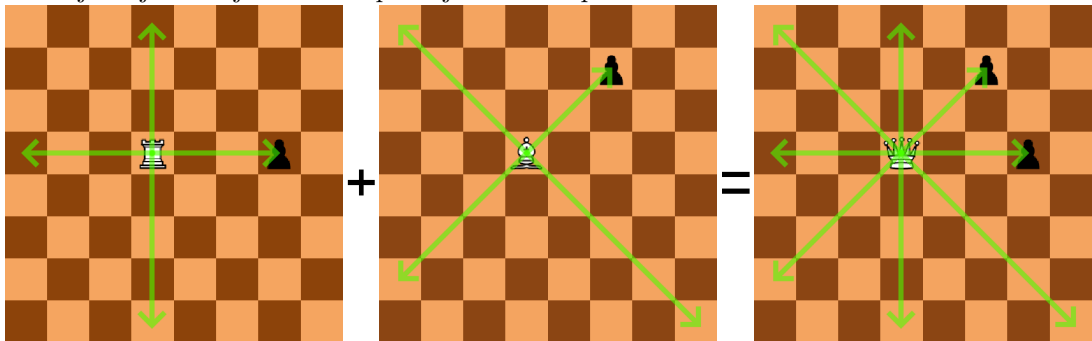
- **isValidBishopMove:**

- Provjerava valjanost poteza lovca, koji se može kretati dijagonalno bilo kojim brojem polja, pod uvjetom da su sva polja na putu prazna.
- Ako je putanja slobodna, potez je valjan.



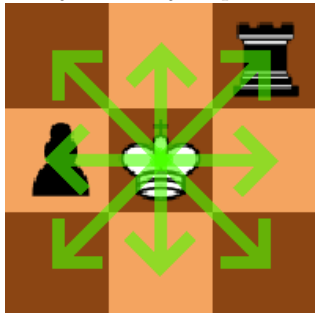
- **isValidQueenMove:**

- Provjerava valjanost poteza kraljice, koja se može kretati kao top (horizontalno ili vertikalno) ili kao lovca (dijagonalno).
- Potez je valjan ako je slobodna putanja kao za top ili lovca.



- **isValidKingMove:**

- Provjerava valjanost poteza kralja, koji se može kretati jedno polje u bilo kojem smjeru (horizontalno, vertikalno ili dijagonalno).
- Provjerava se je li potez unutar jednog polja.



10. Pomoćne funkcije:

- **isOccupiedByOpponent:**

- Provjerava je li polje zauzeto protivničkom figurom.
- Ako je figura na polju različite boje od trenutne figure, vraća **True**, inače **False**.

- **pathIsClear:**

- Provjerava je li putanja kretanja čista, što znači da nema drugih figura na putu između početne i ciljne pozicije.
- Izračunava korake (**deltaX**, **deltaY**) potrebne za kretanje od početne do ciljne pozicije i provjerava svako polje na putu da li je prazno (**Empty**).
- Koristi funkcije **iterate** za generiranje koordinata na putu i **takeWhile** za iteraciju do ciljne pozicije.

Namjena Modula

Modul `ChessLogic.hs` služi kao jezgra logike igre šaha. Njegove funkcionalnosti omogućavaju: - Validaciju poteza i osiguranje da su svi potezi unutar pravila igre. - Ažuriranje stanja ploče na temelju poteza igrača. - Upravljanje izmjenom poteza između igrača i praćenje trenutnog stanja igre.

Ovaj modul je ključan za osiguravanje pravilnog odvijanja igre i za implementaciju osnovnih pravila šaha. Bez njega, igra ne bi mogla pravilno funkcionirati niti bi se mogla osigurati ispravnost poteza.

PlayerInput

```
Enter command for White (e.g., 'e2e4' or 'pb' to print board):
e2e4
Enter command for Black (e.g., 'e2e4' or 'pb' to print board):
a2a3
Invalid command or move. Please try again.
Enter command for Black (e.g., 'e2e4' or 'pb' to print board):
apple
Invalid command or move. Please try again.
- Invalid command format. Use format 'fromSquare toSquare', e.g., 'e2e4'.
Enter command for Black (e.g., 'e2e4' or 'pb' to print board):
```

```
Enter command for Black (e.g., 'e2e4' or 'pb' to print board):
pb
-----
 8 | r  n  b  q  k  b  n  r |
 7 | p  p  p  p  p  p  p  p |
 6 | .  .  .  .  .  .  .  . |
 5 | .  .  .  .  .  .  .  . |
 4 | .  .  .  .  P  .  .  . |
 3 | .  .  .  .  .  .  .  . |
 2 | P  P  P  P  .  P  P  P |
 1 | R  N  B  Q  K  B  N  R |
-----
   a  b  c  d  e  f  g  h
```

terminal_inputs

Modul `PlayerInput.hs` u projektu za šahovsku igru odgovoran je za rukovanje unosom igrača putem terminala. Konkretno, modul pruža funkcionalnosti za:

1. **Petlju za Unos Korisnika:**
 - Implementira petlju koja kontinuirano čeka na unos igrača. Ova petlja omogućava igračima da unesu svoje poteze koristeći terminal.
2. **Validaciju i Obradu Unosa:**
 - Validira unos igrača kako bi osigurala da je u ispravnom formatu (npr. "e2e4" za pomicanje figure s polja e2 na e4).
 - Parsira uneseni potez i ažurira stanje šahovske ploče ako je potez valjan.
3. **Ispis Šahovske Ploče:**
 - Pruža funkcionalnost za ispis trenutnog stanja šahovske ploče na terminal, omogućujući igračima da vide aktualno stanje igre.
4. **Upravljanje Izmjenom Poteza:**
 - Nakon svakog validnog poteza, mijenja boju igrača koji je na potezu, osiguravajući pravilnu izmjenu između bijelog i crnog igrača.

```
module PlayerInput (terminalInputLoop) where

-- Uvoz potrebnih modula
import Data.IORef (IORef, atomicWriteIORef, readIORef) -- IORef za rad s promjenjivim
  ↳ referencama u IO
import System.IO (hFlush, stdout) -- hFlush i stdout za ispis i flushanje izlaza
import Control.Monad (forever, unless) -- forever za beskonačne petlje, unless kao
  ↳ kontrola toka
import ChessLogic (makeMove, isValidMove, switchColor) -- Uvoz funkcija iz ChessLogic
  ↳ modula
```

```

import Chessboard (Chessboard, pieceAt) -- Uvoz tipa Chessboard i funkcije pieceAt iz
↳ Chessboard modula
import ChessPieces -- Uvoz definicija figura i boja iz ChessPieces modula

-- Glavna petlja za unos s terminala
terminalInputLoop :: IORef Chessboard -> IORef Color -> IO ()
terminalInputLoop boardRef colorRef = forever $ do -- Beskonačna petlja
  currentColor <- readIORef colorRef -- Čitanje trenutne boje igrača
  putStrLn $ "Enter command for " ++ show currentColor ++ " (e.g., 'e2e4' or 'pb' to
↳ print board): " -- Ispis poruke za unos
  hFlush stdout -- Osigurava da se poruka odmah prikaže
  command <- getLine -- Čitanje unosa igrača
  board <- readIORef boardRef -- Čitanje trenutnog stanja ploče
  case command of
    "pb" -> printBoard board -- Ako je unos "pb", ispiši ploču
    - -> do
      let updatedBoard = case parseCommand command of -- Parsiranje unosa
        Just (from, to) -> makeMove currentColor from to board -- Ako je unos
↳ valjan, pokušaj obaviti potez
        Nothing -> Nothing -- Ako nije valjan, vrati Nothing
      case updatedBoard of
        Just newBoard -> do
          atomicWriteIORef boardRef newBoard -- Ažuriraj ploču s novim stanjem
          atomicWriteIORef colorRef (switchColor currentColor) -- Promijeni trenutnog
↳ igrača
        Nothing -> do
          putStrLn "Invalid command or move. Please try again." -- Ako je potez
↳ nevaljan, ispiši poruku
          case parseCommand command of
            Nothing ->
              putStrLn " - Invalid command format. Use format 'fromSquare toSquare',
↳ e.g., 'e2e4'." -- Ako je format nevaljan, ispiši poruku
            Just (from, to) -> do
              let pieceAtFrom = pieceAt board from
              case pieceAtFrom of
                Nothing ->
                  putStrLn " - No piece at the specified 'from' square." -- Ako nema
↳ figure na početnoj poziciji, ispiši poruku
                Just piece -> do
                  let isValid = isValidMove board from to
                  unless isValid $
                    putStrLn " - Invalid move according to the rules of chess." -- Ako
↳ potez nije valjan, ispiši poruku

-- Funkcija za ispis cijele šahovske ploče
printBoard :: Chessboard -> IO ()
printBoard board = do
  putStrLn " -----"
  mapM_ \(rank, row) -> putStrLn $ show rank ++ " |" ++ showRow row ++ "|" (reverse
↳ numberedRows) -- Ispis redaka ploče
  putStrLn " -----"
  putStrLn " a b c d e f g h"

where

```

```

    numberedRows = zip [1..8] board -- Dodjeljivanje brojeva redovima ploče
    showRow :: [Square] -> String
    showRow row = unwords $ map showPiece row -- Ispis redaka ploče

-- Funkcija za prikaz pojedine figure ili praznog polja
showPiece :: Square -> String
showPiece (Occupied piece) = showPiece' piece -- Ako je polje zauzeto, prikaži figuru
showPiece Empty = "." -- Ako je polje prazno, prikaži točku

-- Funkcija za prikaz pojedine figure
showPiece' :: Piece -> String
showPiece' (Pawn White) = "P"
showPiece' (Rook White) = "R"
showPiece' (Knight White) = "N"
showPiece' (Bishop White) = "B"
showPiece' (Queen White) = "Q"
showPiece' (King White) = "K"
showPiece' (Pawn Black) = "p"
showPiece' (Rook Black) = "r"
showPiece' (Knight Black) = "n"
showPiece' (Bishop Black) = "b"
showPiece' (Queen Black) = "q"
showPiece' (King Black) = "k"

-- Funkcija za parsiranje naredbe poput "e2e4" u parove koordinata
parseCommand :: String -> Maybe ((Int, Int), (Int, Int))
parseCommand "pb" = Just ((0, 0), (7, 7)) -- Ispis cijele ploče
parseCommand command
  | length command == 4 =
    (,) <$> parseSquare (take 2 command) <*> parseSquare (drop 2 command) --
  ↳ Parsiranje početne i ciljne pozicije
  | otherwise = Nothing -- Ako je format nevaljan, vrati Nothing

-- Funkcija za parsiranje polja poput "e2" u par koordinata
parseSquare :: String -> Maybe (Int, Int)
parseSquare [file, rank]
  | file `elem` ['a'..'h'] && rank `elem` ['1'..'8'] = Just (fileToInt file, rankToInt
  ↳ rank) -- Parsiranje datoteke i ranga
  | otherwise = Nothing -- Ako su datoteka ili rang nevaljani, vrati Nothing
where
  fileToInt c = fromEnum c - fromEnum 'a' -- Konverzija slova u broj
  rankToInt c = fromEnum c - fromEnum '1' -- Konverzija broja u broj
parseSquare _ = Nothing -- Ako format nije "xy", vrati Nothing

```

1. Deklaracije Modula:

- Modul `PlayerInput` definira funkciju `terminalInputLoop` koja će biti izvožena.

2. Uvozi:

- `Data.IORef`: Koristi se za rad s promjenjivim referencama (`IORef`), koje omogućuju dijeljenje i modifikaciju stanja između različitih dijelova programa.
- `System.IO`: Koristi se za rad s unosom i izlazom putem terminala, uključujući ispis i osvježavanje.
- `Control.Monad`: Pruža pomoćne funkcije `forever` (za beskonačne petlje) i `unless` (za uvjetne provjere).
- `ChessLogic`: Uvozi funkcije koje upravljaju logikom igre šaha, uključujući `makeMove`, `isValidMove` i `switchColor`.

- **Chessboard:** Uvozi definicije i funkcije za rad sa šahovskom pločom.
 - **ChessPieces:** Uvozi definicije šahovskih figura i njihovih boja.
3. **Unos Poteza Putem Terminala:**
- **terminalInputLoop**
 - **Opis:** Beskonačna petlja koja upravlja unosom igrača putem terminala.
 - **Rad:**
 - Čita trenutnu boju igrača iz **IORef**.
 - Prikazuje prompt za unos naredbi.
 - Čeka unos naredbe od igrača.
 - Čita trenutnu ploču iz **IORef**.
 - Obrada naredbi:
 - * Ako je naredba “pb”, ispisuje ploču.
 - * Inače, pokušava parsirati naredbu i izvršiti potez.
 - Ažurira ploču i boju igrača ako je potez valjan, inače prikazuje poruku o pogrešci.
4. **Ispis Trenutnog Stanja:**
- **printBoard**
 - **Opis:** Ispisuje trenutno stanje šahovske ploče na terminal.
 - **Rad:**
 - Ispisuje gornju granicu ploče.
 - Ispisuje svaki red ploče s odgovarajućim oznakama redova.
 - Ispisuje donju granicu ploče i oznake stupaca.

Funkcije za Prikaz Figura

5. **showPiece:**
- **Opis:** Vraća tekstualnu reprezentaciju figure ili praznog polja.
 - **Rad:**
 - Ako je polje zauzeto, poziva **showPiece'** za prikaz figure.
 - Ako je polje prazno, vraća točku (“.”).
6. **showPiece':**
- **Opis:** Vraća tekstualnu reprezentaciju pojedine figure.
 - **Rad:**
 - Koristi različite znakove za različite figure, ovisno o njihovoj boji.

Funkcije za Parsiranje Naredbi

7. **parseCommand:**
- **Opis:** Parsira naredbu poput “e2e4” u par koordinata.
 - **Rad:**
 - Ako je naredba “pb”, vraća par koordinata koji označava ispis ploče.
 - Ako je naredba duljine 4 znaka, pokušava parsirati početnu i ciljnu poziciju.
 - Ako naredba nije valjana, vraća **Nothing**.
8. **parseSquare:**
- **Opis:** Parsira polje poput “e2” u koordinatni par.
 - **Rad:**
 - Provjerava je li unos valjan (polje unutar granica ploče).
 - Konvertira slovo stupca i broj reda u odgovarajuće indekse (0-7).

Namjena Modula

Modul **PlayerInput.hs** omogućava interaktivnost igre, omogućujući igračima unos poteza i praćenje stanja igre putem terminala. Njegove funkcionalnosti uključuju:

- Kontinuirano praćenje unosa igrača.
- Validaciju i obradu poteza.

- Ažuriranje stanja šahovske ploče.
- Prikaz trenutnog stanja ploče.

Ovaj modul je ključan za omogućavanje dinamične interakcije između igrača i igre, osiguravajući da igra teče glatko i da igrači mogu lako unositi i vidjeti svoje poteze.

Control.Monad (forever, unless)

- **forever:**
 - **Opis:** `forever` uzima akciju i ponavlja je beskonačno.
 - **Primjer u šahu:** U `terminalInputLoop` funkciji se koristi `forever` da bi se kontinuirano prihvatao unos od igrača, omogućujući beskonačnu petlju za unos naredbi.
 - **Primjer:**

```
terminalInputLoop boardRef colorRef = forever $ do
    -- Kod za unos naredbi od igrača ...
```
- **unless:**
 - **Opis:** `unless` uzima uvjet i akciju, i izvršava akciju samo ako je uvjet `False`.
 - **Primjer u šahu:** U funkciji za validaciju poteza koristi se `unless` za ispis poruke o nevažećem potezu samo ako potez nije valjan.
 - **Primjer:**

```
unless isValid $
    putStrLn " - Invalid move according to the rules of chess."
```

Data.IORef (IORef, atomicWriteIORef, readIORef)

- **IORef:**
 - **Opis:** `IORef` je referenca koja omogućuje mutabilne promjene stanja u Haskellu, unutar IO monada.
 - **Primjer u šahu:** `IORef` se koristi za čuvanje stanja šahovske ploče i trenutne boje igrača.
 - **Primjer:**

```
boardRef <- newIORef initialChessboard
colorRef <- newIORef White
```
- **atomicWriteIORef:**
 - **Opis:** `atomicWriteIORef` atomarno zapisuje novu vrijednost u `IORef`, osiguravajući da se promjena dogodi bez ikakvih međuprostornih stanja.
 - **Primjer u šahu:** Kada se napravi validan potez, ažurira se stanje ploče i boja igrača atomarno.
 - **Primjer:**

```
atomicWriteIORef boardRef newBoard
atomicWriteIORef colorRef (switchColor currentColor)
```
- **readIORef:**
 - **Opis:** `readIORef` čita trenutnu vrijednost iz `IORef`.
 - **Primjer u šahu:** U `terminalInputLoop` funkciji, čita se trenutna boja igrača i stanje ploče.
 - **Primjer:**

```
currentColor <- readIORef colorRef
board <- readIORef boardRef
```

System.IO (hFlush, stdout)

- **hFlush:**
 - **Opis:** `hFlush` ispraznjava međuspremnik (buffer) za dani `Handle` (obično `stdout`), osiguravajući da se svi podaci odmah pošalju na izlaz.
 - **Primjer u šahu:** Nakon prikaza prompta za unos, `hFlush` se koristi da bi se osiguralo da prompt bude odmah vidljiv korisniku.
 - **Primjer:**

```
hFlush stdout
```

- **stdout:**
 - **Opis:** `stdout` je standardni izlaz, obično terminal ili konzola, gdje se ispisuju podaci.
 - **Primjer u šahu:** Koristi se zajedno s `putStrLn` i `hFlush` za prikazivanje prompta za unos igrača.
 - **Primjer:**

```
putStrLn $ "Enter command for " ++ show currentColor ++ " (e.g., 'e2e4' or 'pb'
↪ to print board): "
hFlush stdout
```


Funkcijsko programiranje u razvoju šahovske igre

Isticanje upotrebe nemjenjivih struktura podataka i čistih funkcija u razvoju igre

- **Nemjenjive strukture podataka:**
 - U Haskell-u, šahovska ploča je definirana kao nemjenjiva struktura podataka (immutable). Na primjer, `initialChessboard` je nemjenjiva početna ploča igre
 - Promjene stanja ploče rezultiraju novim objektima ploče, umjesto izmjene postojećih, čime se izbjegava neželjene nuspojave.
- **Čiste funkcije:**
 - `makeMove` je čista funkcija koja ne mijenja vanjsko stanje, već vraća novo stanje šahovske ploče

Demonstracija kako funkcijski pristup pojednostavljuje upravljanje složenim interakcijama unutar igre

- **Jednostavnost upravljanja stanjem:**
 - `isEmpty` funkcija je čista i jednostavna funkcija koja provjerava je li određeno polje prazno
 - Stanje igre se može jednostavno pratiti kroz promjene stanja referenci (`IORef`), koje sadrže trenutno stanje
- **Deklarativan stil:**
 - U funkcijskom programiranju, pravila igre se mogu definirati deklarativno. Na primjer, validacija poteza `isValidMove`

Usporedba s imperativnim i objektno-orijentiranim pristupima u razvoju softvera za igre

- **Imperativni pristup:**
 - U imperativnom programiranju, promjene stanja su česte i često nepredvidljive. Na primjer, kod u jeziku C#:

```
public void makeMove(Board board, int fromX, int fromY, int toX, int toY) {
    Piece piece = board.Squares[fromY, fromX];
    board.Squares[toY, toX] = piece;
    board.Squares[fromY, fromX] = null;
}
```
 - Ovakav kod može biti složen za praćenje i održavanje zbog izravnog upravljanja memorijom i stanjima.
- **Objektno-orijentirani pristup:**
 - U objektno-orijentiranom programiranju, stanje igre je kapsulirano unutar objekata. Na primjer, kod u jeziku Java:

```
class ChessGame {
    private Board board;
    private Player currentPlayer;

    public void makeMove(Position from, Position to) {
        Piece piece = board.getPiece(from);
        board.setPiece(to, piece);
        board.setPiece(from, null);
        switchPlayer();
    }
}
```
 - Iako je kod organiziraniji, nasljeđivanje i mutabilnost mogu uzrokovati dodatne komplikacije.
- **Funkcijski pristup:**
 - U funkcijskom programiranju, koristimo nemjenjive strukture i čiste funkcije

Analiza prednosti funkcijskog pristupa u održivosti i skalabilnosti logike igre

- **Održivost:**

- Korištenje nemjenjivih struktura i čistih funkcija smanjuje mogućnost grešaka i olakšava razumijevanje koda. Na primjer, `isValidMove` funkcija je izolirana i lako testirana
- **Skalabilnost:**
 - Funkcijski pristup omogućava jednostavno paraleliziranje operacija zbog nemjenjivosti stanja. Na primjer, funkcija `env2Pic` koristi čiste funkcije za prikaz šahovske ploče
- **Testiranje i verifikacija:**
 - Testiranje čistih funkcija je jednostavnije jer su determinističke.
 - Ovo povećava pouzdanost i omogućava jednostavniju detekciju i ispravku grešaka.

Korišteni resursi

U razvoju ovog projekta korišteni su brojni resursi koji su značajno doprinijeli njegovoj realizaciji. Slijedi detaljan pregled korištenih resursa:

- **Gloss biblioteka za grafiku:**
 - **Opis:** Gloss je Haskellova biblioteka dizajnirana za jednostavno i brzo stvaranje 2D grafike, animacija i simulacija. Pruža intuitivno API sučelje koje omogućava brzo crtanje osnovnih oblika, teksta i slika.
 - **Link:** Gloss na Hackage-u
 - **Upotreba u projektu:** Gloss biblioteka je ključna za crtanje šahovske ploče i figura. Omogućila je jednostavno stvaranje i ažuriranje grafičkog prikaza igre, te implementaciju animacija za pomicanje figura.
- **Haskell dokumentacija:**
 - **Opis:** Službena Haskell dokumentacija pruža detaljan pregled jezika, njegovih značajki, te standardnih biblioteka. Sadrži vodiče, referentne materijale i primjere koji pomažu programerima u učenju i primjeni Haskell.
 - **Link:** Haskell dokumentacija
 - **Upotreba u projektu:** Dokumentacija je korištena za razumijevanje osnovnih i naprednih koncepta Haskell jezika, kao što su rad s tipovima, IO operacije, rad s listama i funkcionalno programiranje općenito. Također je pružila uvid u korištenje specifičnih biblioteka i modula.
- **Različiti online resursi i tutorijali za funkcionalno programiranje i Haskell:**
 - **Opis:** Online resursi uključuju blogove, tutorijale, forume i video predavanja koji pokrivaju širok raspon tema vezanih uz funkcionalno programiranje i Haskell. Oni su izvor praktičnih savjeta, primjera koda i najbolje prakse.
 - **Primjeri:**
 - * **Learn You a Haskell for Great Good!:** Popularni online vodič i knjiga za učenje Haskell kroz praktične primjere i humorističan pristup. Learn You a Haskell
 - * **Hoogle:** Online pretraživač za Haskell API-je koji omogućava brzo pronalaženje funkcija i modula prema tipovima i nazivima. Hoogle
 - * **Stack Overflow:** Forum za programere gdje se može naći mnogo odgovora na specifična pitanja o Haskellu i funkcionalnom programiranju. Stack Overflow
 - * **YouTube kanali:** Kanali koji nude video lekcije i tutorijale o Haskellu i funkcionalnom programiranju, poput kanala Philipp Hagenlocher, Beaufort Tek i Derek Banas.
 - **Upotreba u projektu:** Ovi resursi su korišteni za učenje Haskell, rješavanje problema, te za dobivanje inspiracije i ideja za implementaciju različitih značajki igre.
- **Razvojni alati i okruženja:**
 - **Opis:** Različiti alati i okruženja korišteni su za razvoj, testiranje i otklanjanje grešaka u projektu.
 - **GHC (Glasgow Haskell Compiler):** Standardni kompajler za Haskell koji podržava napredne značajke jezika.
 - **Stack:** Alat za upravljanje projektima i paketima u Haskellu, koji olakšava instalaciju ovisnosti i upravljanje verzijama. Stack
 - **VS Code s Haskell ekstenzijama:** Popularni uređivač koda s podrškom za Haskell kroz ekstenziju koja pruža funkcionalnosti poput isticanja sintakse, automatskog dovršavanja koda i integracije s kompajlerom.
 - * Haskell Syntax Highlighting
 - Omogućava isticanje sintakse Haskell koda, prepoznajući ključne riječi, tipove, funkcije i komentare.
 - **Značajke:** Isticanje ključnih riječi Haskell, Poboljšanje čitljivosti koda
 - * Haskell
 - Pruža integriranu podršku za Haskell razvoj u Visual Studio Code, uključujući isticanje sintakse, automatsko dovršavanje koda, provjeru tipova i integraciju s GHCi.
 - **Značajke:** Automatsko dovršavanje koda, Provjera tipova u stvarnom vremenu, Integracija s GHCi, Navigacija kodom
 - * Haskellly

- Poboljšava iskustvo pisanja Haskell koda, uključujući isticanje sintakse, provjeru tipova i automatsko formatiranje koda.
- **Značajke:** Isticanje sintakse, Provjera tipova, Automatsko formatiranje koda, Linting