ИНСТИТУТ ЗА МАТЕМАТИКУ И ИНФОРМАТИКУ

ПРИРОДНО-МАТЕМАТИЧКОГ ФАКУЛТЕТА

УНИВЕРЗИТЕТА У КРАГУЈЕВЦУ

Семинарски рад

Предмет: Логичко и функцијско програмирање

Тема: Реализација игре памћења у програмском језику Haskell

Студент: Марија Андрић, 24/2020

Ментор: Др Татјана Стојановић

Садржај

1. Увод 3
   - О игри 3
   - Правила игре 3
   - О пројекту 3
2. Ток и изглед имплементиране игре 3
   - Почетак 4
   - Ток 4
   - Крај 6
3. Преглед и објашњење кода 7
   - Импортовање библиотека и дефиниција типова 7
   - Дефинисање и мешање елемената низа 8
   - Цртање табле и картица 9
   - Иницијално стање игре 10
   - Логика 10
   - Маин 12
4. Референце 13

1. Увод

○ О игри

Игра памћења, такође позната као "Пронађи парове" или "Игра меморије", је класична игра памћења и концентрације која је популарна међу различитим узрастима.

Историја игре памћења сеже уназад децијама. Прве верзије ове игре датирају још из 16. века, а потичу из Јапана, где су одговарајуће слике насликане на шкољкама знане као asAwase или Kai-awase. Игра се брзо проширила свуда по свету, али онако како је ми знамо данас, у данашњем облику објавио ју је Равенсбургер у фебруару 1959. године.

Постоје више циљева ове игре. Она, између осталог, подстиче играче да развијају своју способност концетрације и памћења. Кроз процес окретања карата и тражења одговарајућих картица, играчи морају да памте положаје различитих карата како би пронашли одговарајуће парове током игре. Поред тога, игра памћења подстиче развој стратегијског размишљања јер су играчи суочени са одлукама о томе које карте могу да окрену и тако траже начине да максимизују своје шансе за проналажење парова уз минималан број окретања.

У својој суштини, игра памћења представља комбинацију забаве и образовања. Њена способност да ангажује играче на емоционалном и интелектуалном нивоу чини је популарним избором за све узрасте.

○ Правила игре

Играч окреће 2 карте. Ако се слике поклапају, играч задржава карте и покушава поново да пронађе нови пар. Ако се карте не поклапају, карте се поново окрећу тако да играч не види шта је на њима. Играч треба да покуша да се сети где је које карте видео како би игру завршио што пре. Крај игре је када играч пронађе све парове карата, тј. када су све карте отворене.

○ О пројекту

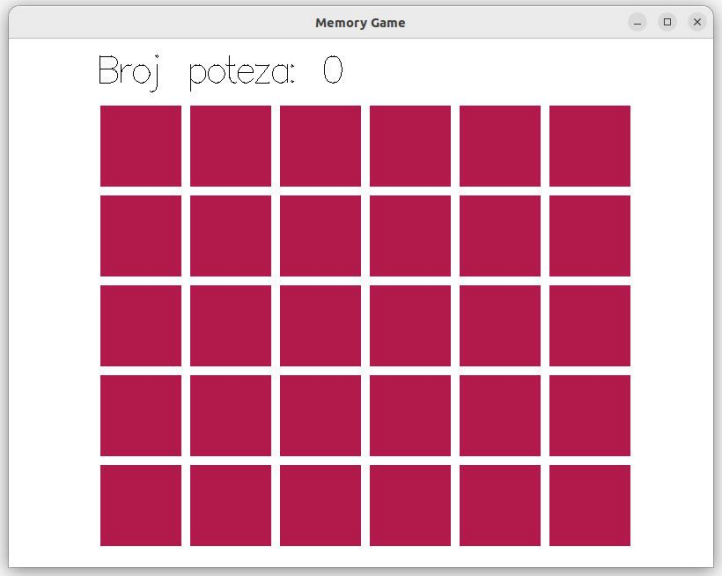
Реализован пројекат обухвата имплементацију претходно описане игре у програмском језику Haskell. Идеја је пренос класичне игре памћења у контекст функционалног програмирања. За реализацију ово пројекта коришћена је графичка библиотека Graphics.Gloss. Она је део Haskell платформе и омогућава цртање различирих типова слика, могућност креирања анимација, рад са бојама и текстуалним елементима и то користећи функционални стил програмирања.

2. Ток и изглед имплементиране игре

Ток игре памћења се базира на последици окретања и тражења карата, са намером да се пронађу парови који се подударају.

Почетак

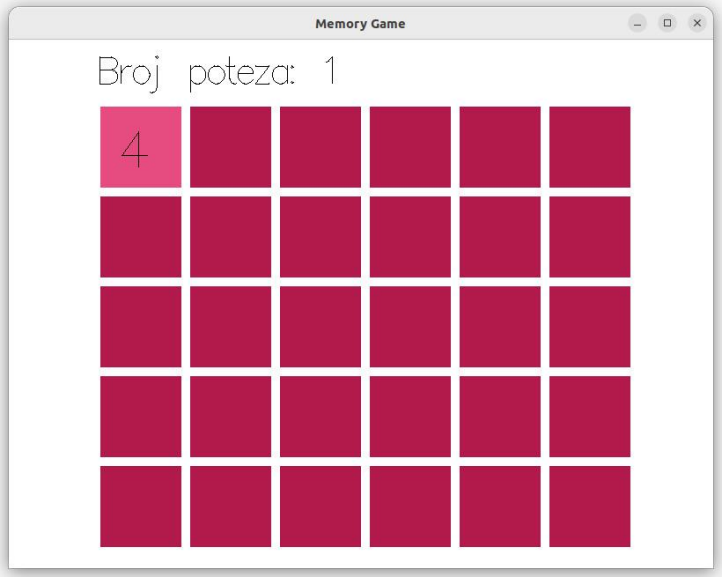
Игра почиње тако што се на екрану налази табла са картицама која има 5 редова и 6 колона. На почетку игре, картице су празне и немају никакве бројеве на себи. Изнад картица, налази се бројач потеза који је на почетку игре једнак нули.



Игра памћења 1 - Почетни изглед табле

Ток

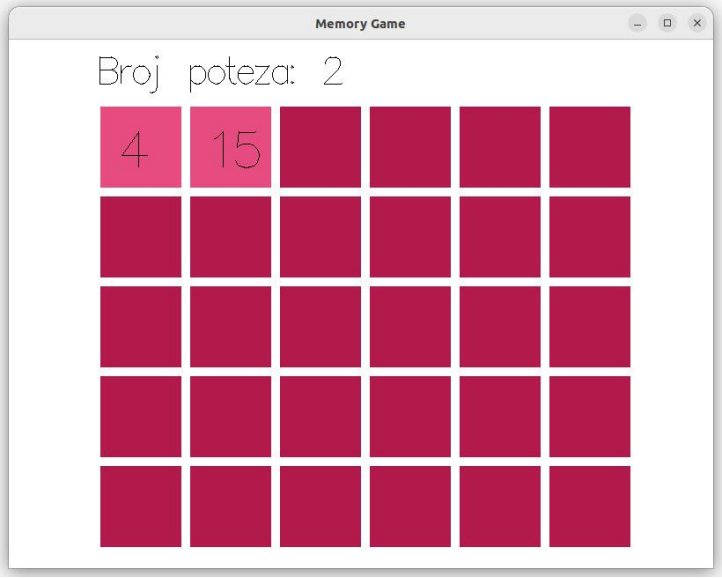
Игра почиње тако што играч може да кликне на обојено поље које представља картицу. Кликом на карту, она се „окреће“, картице више није празна, сада садржи број. Играч треба да нађе још једну карту са истим бројем како би картице остале „окренуте“. Кликом на било коју „затворену“ карту, повећава се број потеза за један. Циљ је да број потеза буде што мањи.



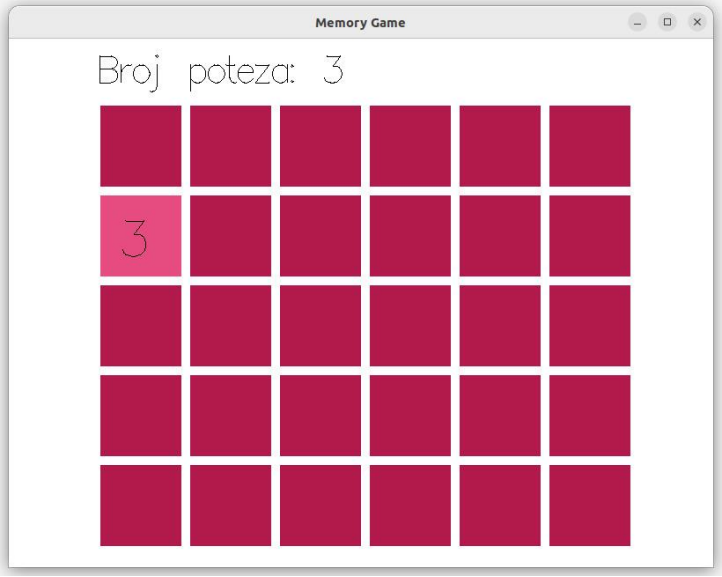
Игра памћења 2 - Клик на једну картицу

Постоје две могућности када се „окрену“ две карте.

Прва могућност је да су пронађене карте различите. У том случају, оне ће бити „окренуте“, све док се не деси нови клик на неку другу затворену карту.

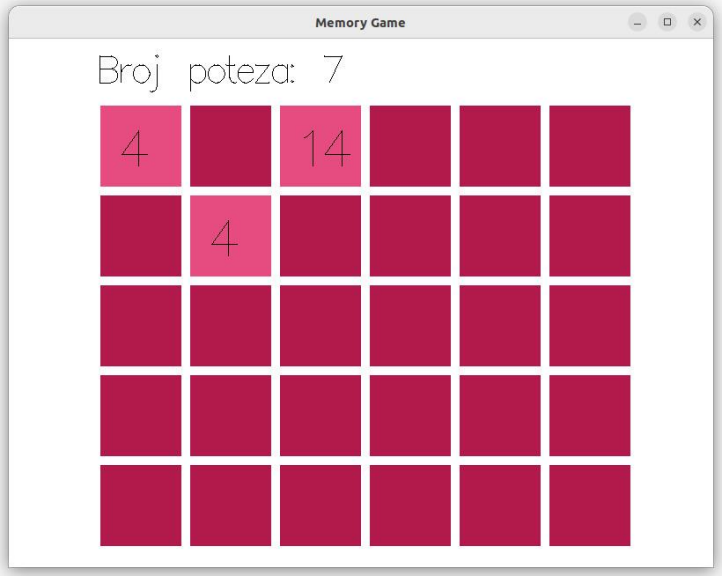


Игра памћења 3 - Две различите окренуте карте



Игра памћења 4 - Клик на другу карту

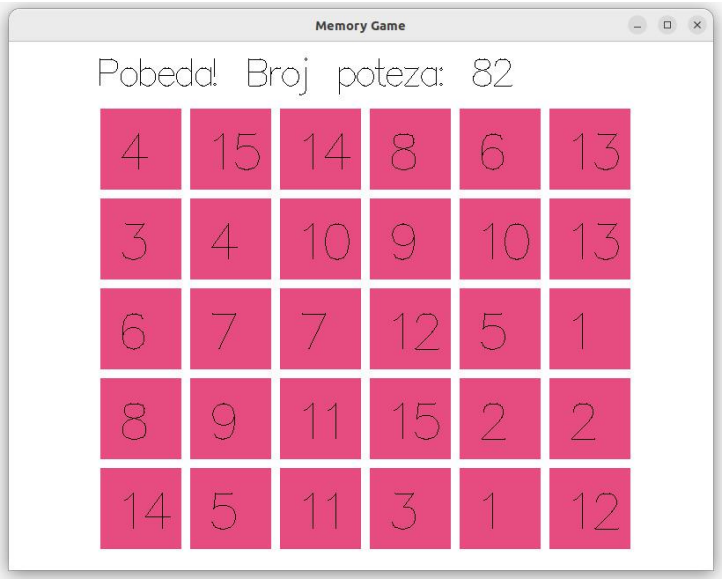
Друга могућност обухвата две „окренуте“ карте које имају исти број. У том случају, те две карте ће бити отворене до краја игре. Кликом на другу карту, оне ће остати отворене, а играч ће моћи да налази нови пар карата.



Игра памћења 5 - Приказ клика на другу карту након проналаска две исте карте

- ## Крај

Крај игре је кад играч пронађе све парове карата. У том случају, информација о победи ће му се приказати на врху екрана.



Игра памћења 6 - Крај игре и промена текста

3. Преглед и објашњење кода

У наставку се налази објашњење имплементације кода.

○ Импортовање библиотека и дефиниција типова

На самом почетку, импортоване су библиотеке неопходне за покретање овог пројекта. Одмах испод њих, дефинисани су типови који ће се користити кроз даљи код.

Тип „Card“ претставља једну картицу која се састоји од:

1. visina - висина картице
2. širina - ширина картице
3. x - који је ред у оквиру табле
4. y- колона у оквиру табле
5. randomNumber - број који се приказује када се картица „отвори“
6. isFlip - поље које даје одговор на питање да ли је картица тренутно „отворена“?
7. isFound - поље које даје одговор на питање да ли је пронађен пар те картице?

Тип „GameState“ представља тренутно стање игре и састоји се од:

1. cards - скуп свих карата
2. selectedCardIndices - индекс карата у листи cards које су „отворене“
3. moveCount - број потеза

```haskell
import Graphics.Gloss
import System.Random
import System.IO.Unsafe (unsafePerformIO)
import Graphics.Gloss.Interface.Pure.Game
import Debug.Trace


data Card = Card
  { visina :: Float
  , sirina :: Float
  , x :: Int
  , y :: Int
  , randomNumber :: Int
  , isFlip :: Bool
  , isFound :: Bool
  }


data GameState = GameState
  { cards :: [Card]
  , selectedCardIndices :: [Int]
  , moveCount :: Int
  }

```

Игра памћења 7 - имиортоване библиотеке и дефинисање типова

```haskell

windowWidth, windowHeight :: Int
windowWidth = 800
windowHeight = 600


boardWidth, boardHeight :: Float
boardWidth = 500
boardHeight = 500


cardWidth, cardHeight :: Float
cardWidth = (boardWidth - cardSpacing * 4) / 5
cardHeight = (boardHeight - cardSpacing * 4) / 5


cardSpacing :: Float
cardSpacing = 10

```

Игра памћења 8 – Дефинисање ширине и висине екрана, табле са картицама, картица и размака између картица

○ Дефинисање и мешање елемената низа

Са дефинисаним низом који садржи бројеве од 1 до 15, који се понављају два пута, позива се функција shuffle која има задатак да промеша бројеве у низу. Ова функција користи NewStdGen како би добила нови генератор случајних бројева. Низ и генератор случајних бројева се шаље функцији shuffle' која генерише насумични индекс уз помоћ функције RandomGen и користи га да „извуче“ елемент из листе, након чега се он брише функцијом removeAt.

```haskell
numbers :: [Int]
numbers = [1..15] ++ [1..15]

shuffle :: [a] -> IO [a]
shuffle xs = do
    gen <- newStdGen
    return $ shuffle' gen xs


shuffle' :: RandomGen g => g -> [a] -> [a]
shuffle' _ [] = []
shuffle' gen xs = x : shuffle' gen' xs'
  where
    (index, gen') = randomR (0, length xs - 1) gen
    (x, xs') = removeAt index xs


removeAt :: Int -> [a] -> (a, [a])
removeAt n xs = (xs !! n, take n xs ++ drop (n + 1) xs)


getShuffledNumbers :: [Int]
getShuffledNumbers = unsafePerformIO $ shuffle numbers


```

Игра памћења 9 – Низ са измешаним бројевима

Овде се **unsafePerformIO** користи како би се функција shuffle третирала као „чиста“ функција иако заправо користи „И/О“.

○ Цртање табле и картица

Функције drawBoard и drawCard враћа тип **Picture**, што значи да су задужене за констрисање слика, односно цртежа који се може приказати на екрану. Функција drawBoard прихвата списак картица и број потеза. Пролазећи кроз низ картица, позива методу drawCard која исцртава карте на табли. Функција **translate** је функција библиотеке Graphics.Gloss која омогућава померање слика односно цртежа. Функција **scale** исто припада овој библиотеци и она омогућава промену величине цртежа, а **makeColor** омогућава креирање боја по жељи. Функција **fromIntegral** се користи за конвертовање броја у број са покретним зарезом.
Функција **pictures** комбинује више графичких елемената у једну сцену која ће бити приказана на екрану, у овом случају текст и картице.

```
68
69
70  drawCard :: Float -> Float -> Int -> Int -> Int -> Bool -> Bool -> Picture
71  drawCard x y corx cory memoryNumber isFlip isFound= translate (x - cardWidth / 2) (y - cardHeight / 2) $
72    pictures [ cardShape, cardText ]
73    where
74      cardShape = if isFlip || isFound
75        then color (makeColor 0.9 0.3 0.5 1.0) $ rectangleSolid cardWidth cardHeight
76        else color (makeColor 0.7 0.1 0.3 1.0) $ rectangleSolid cardWidth cardHeight
77
78      cardText = if isFlip || isFound
79        then translate (-cardWidth / 4) (-cardHeight / 4) $ scale 0.4 0.4 $ color black $ text (show memoryNumber)
80        else blank
81
82
83  drawBoard :: [Card] -> Int -> Picture
84  drawBoard cardList moveCount=
85    pictures
86      [ translate (-300) 250 $ scale 0.3 0.3 $ color black $ text gameTitle
87      , pictures [ drawCard
88                     (fromIntegral (x card) * (cardWidth + cardSpacing) + cardWidth / 2 -250)
89                     (fromIntegral (y card) * (cardHeight + cardSpacing) + cardHeight / 2 -230)
90                     (x card)
91                     (y card)
92                     (randomNumber card)
93                     (isFlip card)
94                     (isFound card)
95                 | card <- cardList
96                 ]
97      ]
98      where
99      gameTitle = if areAllCardsFlip cardList
100                   then "Pobeda! Broj poteza: "++ show (moveCount)
101                   else "Broj poteza: " ++ show (moveCount)
102
103
104 areAllCardsFlip :: [Card] -> Bool
105 areAllCardsFlip cards =
106   let numFound = length (filter isFlip cards)
107   in numFound == length cards
108
```

Игра памћења 10 - Графика игрице

Постоји и функција areAllCardsFlip која проверава да ли су све картице отворене и ако је то тачно, у функцији drawBoard мења се текст који се налази изнад табле за играње.

○ Иницијално стање игре

У функцији на слици „Игра памћења 11“ поставља се иницијално стање игрице, прави се списак картица, где се између осталог дефинише њихова позиција и стање.

```
108
109
110  initial :: [Card]
111  initial = [Card
112                  (fromIntegral x * (cardWidth + cardSpacing) + cardWidth / 2 -250)
113                  (fromIntegral y * (cardHeight + cardSpacing) + cardHeight / 2 -230)
114                  x
115                  y
116                  (shuffledNumbers !! (x * 5 + y))
117                  False
118                  False
119            | x <- [0 .. 5], y <- [0 .. 4]]
120   where
121    shuffledNumbers = getShuffledNumbers
122
123
```

Игра памћења 11 - Иницијално стање игре

○ Логика

Што се тиче логике, разматраћемо следеће функције:

1. handleEvent - Ова функција узима тренутно стање игре и догађај како би се развила логика игре. Она проверава да ли се догађај односи на клик левог тастера миша, и ако јесте омогућава да се клик обради. Постоје две опције, клик на карту и клик ван ње, па зависно од тога позива функције handleCardClick и handleOtherClick.
2. handleCardClick - Ова функција проверава да ли је карта на коју је неко кликнуо већ „отворена“ или не, и ако није отвара карту. Онда мења читаву листу карата користећи методу updateCardAtIndex како би цела листа била ажурирана. Проверава број селектованих картица и преправља листу индекса селектованих картица на сваки нови клик, али и ажурира тренутно стање игре. Ова функција такође позива и методу compareCards како би проверила да ли се две отворене карте исте, ажурирајући их.
3. handleOtherClick - Ова функција исписује у конзоли да је играч кликнуо ван картице.
4. isMouseOnCard- проверава да ли се на прослеђеним координатама клика налази прослеђена карта.
5. compareCards – За прослеђено стање игрице, узима селектоване индексе и упоређује картице са тим индексима, преправљујући њихово стање и ажурирајући стање игрице.

```haskell
compareCards :: GameState -> GameState
compareCards gameState =
  let selectedIndices = selectedCardIndices gameState
  in if length selectedIndices == 2
     then
       let idx1 : idx2 : _ = selectedIndices
           card1 = cards gameState !! idx1
           card2 = cards gameState !! idx2
           areEqual = randomNumber card1 == randomNumber card2
           updatedCard1 = card1 { isFlip = areEqual, isFound = areEqual }
           updatedCard2 = card2 { isFlip = areEqual, isFound = areEqual }
           updatedCards = updateCardAtIndex idx1 updatedCard1 $ updateCardAtIndex idx2 updatedCard2 (cards gameState)
           newGameState = gameState { cards = updatedCards }
           updatedGameState = if null selectedIndices
                              then closeFlippedUnfound newGameState
                              else newGameState
       in updatedGameState
     else if null selectedIndices
          then closeFlippedUnfound gameState
          else gameState


closeFlippedUnfound :: GameState -> GameState
closeFlippedUnfound gameState =
  let cardsToClose = [idx | (idx, card) <- zip [0..] (cards gameState), isFlip card && not (isFound card)]
      updatedCards = foldr (\idx acc -> updateCardAtIndex idx (cards gameState !! idx) acc) (cards gameState) cardsToClose
  in gameState { cards = updatedCards }


isMouseOnCard :: Float -> Float -> Card -> Bool
isMouseOnCard mouseX mouseY card =
  let cardX = visina card
      cardY = sirina card
  in mouseX >= cardX - cardWidth && mouseX <= cardX
     && mouseY >= cardY - cardHeight && mouseY <= cardY

```

Игра памћења 12

```haskell
handleCardClick :: Int -> GameState -> GameState
handleCardClick idx gameState =
  let clickedCard = cards gameState !! idx
  in if isFound clickedCard || isFlip clickedCard
     then gameState
     else
       let gameStateAfterCompare = compareCards gameState
           updatedCard = clickedCard { isFlip = True }
           updatedCards = updateCardAtIndex idx updatedCard (cards gameStateAfterCompare)
           selectedIndices = selectedCardIndices gameStateAfterCompare
           newSelectedIndices = if length selectedIndices == 2
                                then [idx]
                                else if not (isFlip clickedCard) && length selectedIndices < 2
                                     then idx : selectedIndices
                                     else selectedIndices
           updatedMoveCount = moveCount gameStateAfterCompare + 1
           updatedGameState = gameStateAfterCompare { cards = updatedCards, selectedCardIndices = newSelectedIndices, moveCount = updatedMoveCount }
       in updatedGameState


updateCardAtIndex :: Int -> Card -> [Card] -> [Card]
updateCardAtIndex idx newCard cards = take idx cards ++ [newCard] ++ drop (idx + 1) cards


handleOtherClick :: GameState -> GameState
handleOtherClick gameState = trace "Klik izvan kartica!" gameState


handleEvent :: Event -> GameState -> GameState
handleEvent (EventKey (MouseButton LeftButton) Down _ (mouseX, mouseY)) gameState =
  case cardIndices of
    [idx] -> handleCardClick idx gameState
    _     -> handleOtherClick gameState
  where
    cardIndices = [i | (i, card) <- zip [0..] (cards gameState), isMouseOnCard mouseX mouseY card]
handleEvent _ gameState = gameState


```

Игра памћења 13

○ Маин

Ова функција се извршава приликом покретања програма. Тип И/О означава да се у оквиру функције врши „улазно/излазно“ интераговање, у овом случају графички интерфејс и догађаји попут клика миша. Објашњење позваних функција унутар маин функције:

1. displayMode – Приказ прозора игре
2. backgroundColor - Боја позадине екрана игре
3. frameRate - Брзина освежавања екрана
4. initialState - Поставља иницијално стање игре
5. render - Функција за цртање тренутног стања игре
6. handleEvent - Претходно објашњена функција која управља догађајем
7. update - Функција која се користи за ажурирање стања игре током времена

```
198
199
200  main :: IO ()
201  main = play displayMode backgroundColor frameRate initialState render handleEvent update
202    where
203      prom = initial
204      displayMode = InWindow "Memory Game" (windowWidth, windowHeight) (100, 100)
205      backgroundColor = white
206      frameRate = 60
207      initialState = GameState prom [] 0
208      render gameState = drawBoard (cards gameState) (moveCount gameState)
209      update _ = id
210
211
212
```

Ишра йамћења 14 - Маин функција

4. Референце

Овде се налазе линкови који су релевантни са овим семинарским радом, укључујући линкове који су били од помоћи за реализацију кода:

1. https://board-games-galore.fandom.com/wiki/Memory
2. https://hackage.haskell.org
3. https://mmhaskell.com/blog/2019/3/25/making-a-glossy-game-part-1#google_vignette
4. https://fosdem.org/2023/schedule/event/haskell_2d_animations/
5. https://blog.ocharles.org.uk/posts/2013-12-10-24-days-of-hackage-gloss.html
6. https://hoogle.haskell.org/?hoogle=StdGen
7. https://wiki.haskell.org