Find The Food! :Game In Haskell

Module Physik653

WS 2019-2020

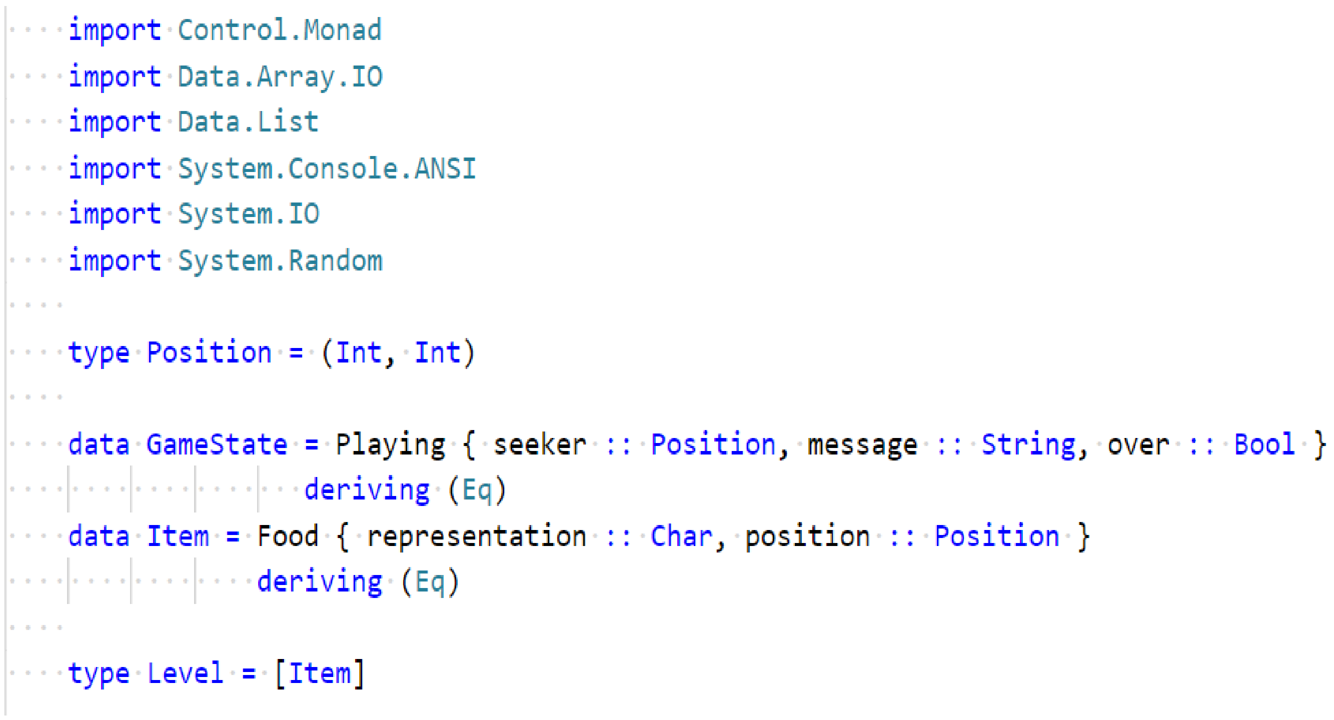
Seminar On Functional Programming Languages For Physicists

-Chelsea John

Concept of the Game:

- Introduce the food and seeker at random position
- Navigate the seeker to the food through user inputs
- When seeker finds the food , Game over !

SETTING THE STAGE:

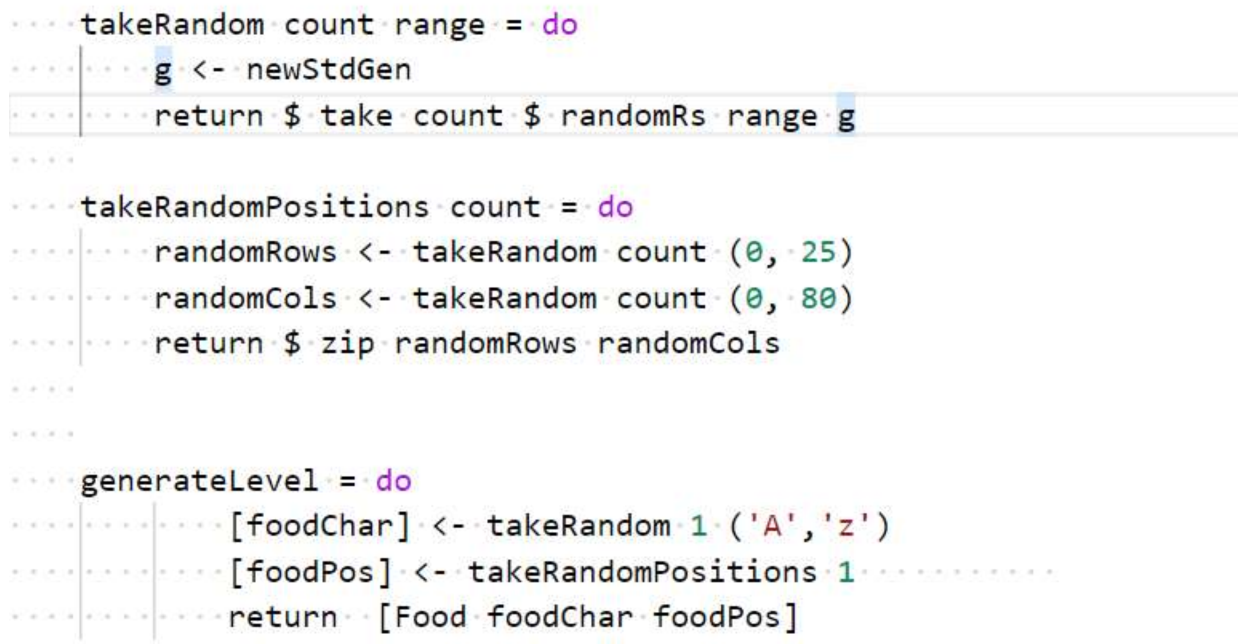


```haskell
    import Control.Monad
    import Data.Array.IO
    import Data.List
    import System.Console.ANSI
    import System.IO
    import System.Random

    type Position = (Int, Int)

    data GameState = Playing { seeker :: Position, message :: String, over :: Bool }
                       deriving (Eq)
    data Item = Food { representation :: Char, position :: Position }
                    deriving (Eq)

    type Level = [Item]
```

Route structure for the game:

- Step 1

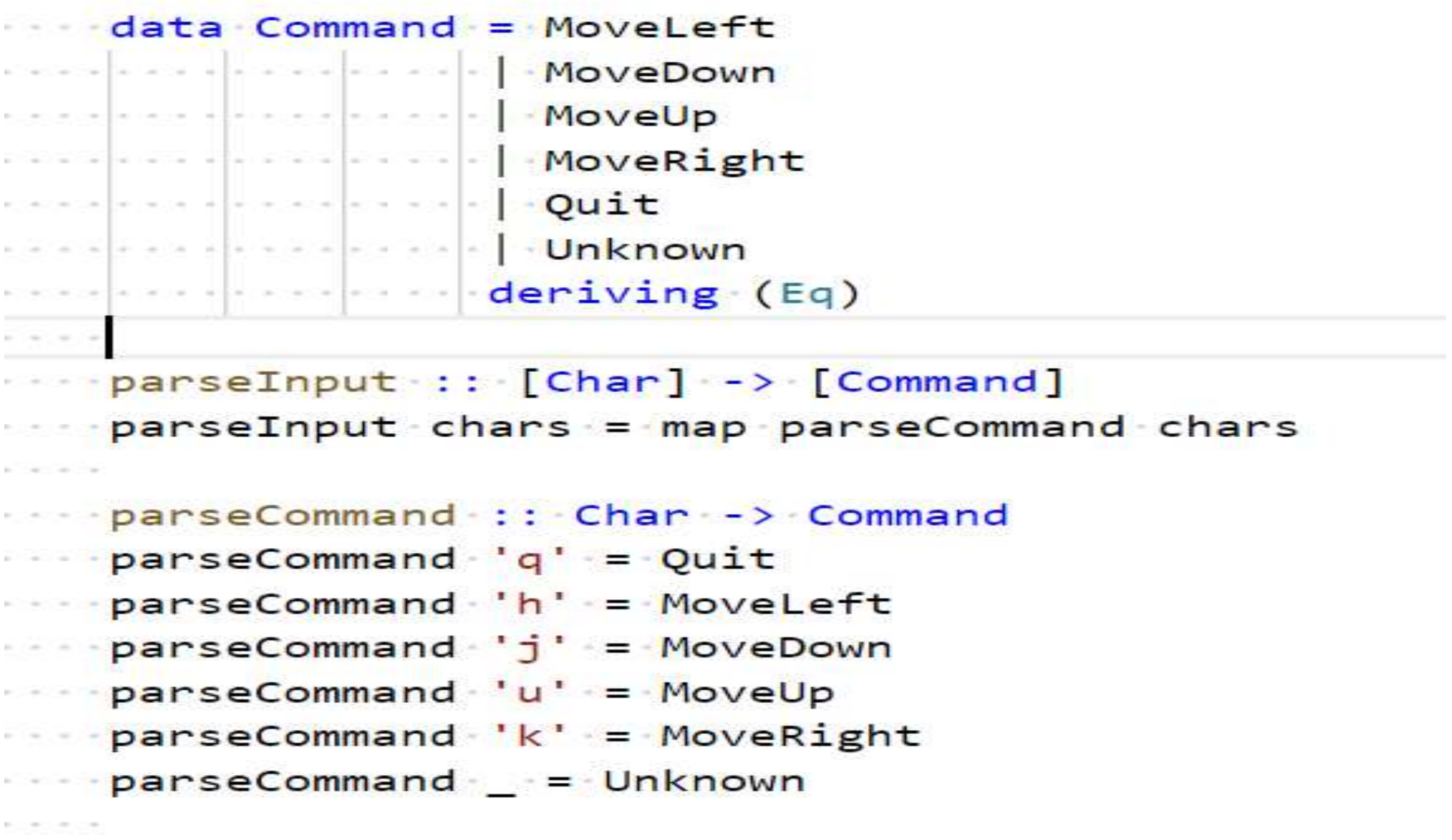
  Generate random positions to place the seeker and food in the beginning of the game



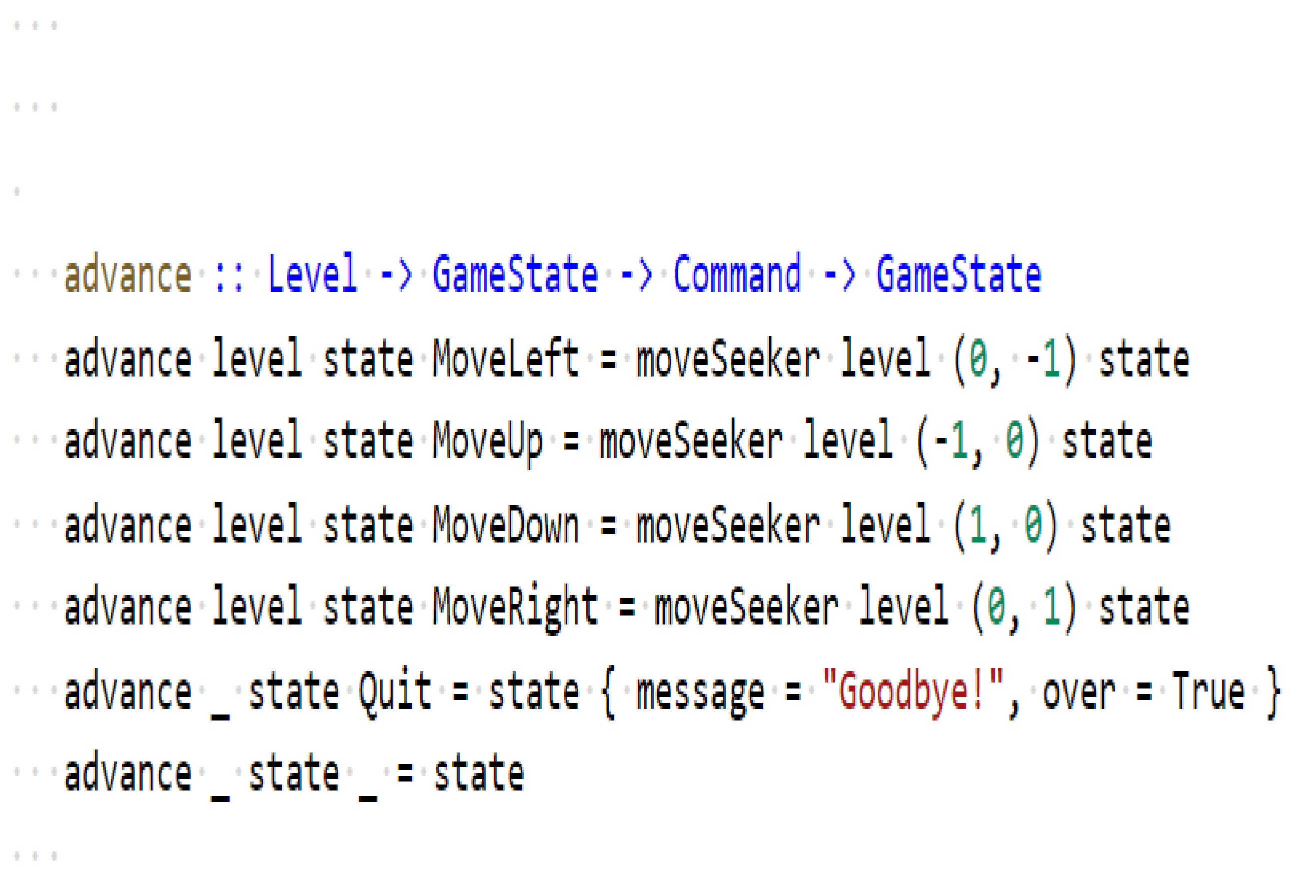
```haskell
    takeRandom count range = do
        g <- newStdGen
        return $ take count $ randomRs range g

    takeRandomPositions count = do
        randomRows <- takeRandom count (0, 25)
        randomCols <- takeRandom count (0, 80)
        return $ zip randomRows randomCols


    generateLevel = do
            [foodChar] <- takeRandom 1 ('A','z')
            [foodPos] <- takeRandomPositions 1
            return  [Food foodChar foodPos]
```

- Step 2

  Define commands and keys(**h,u,j,k**) for the game



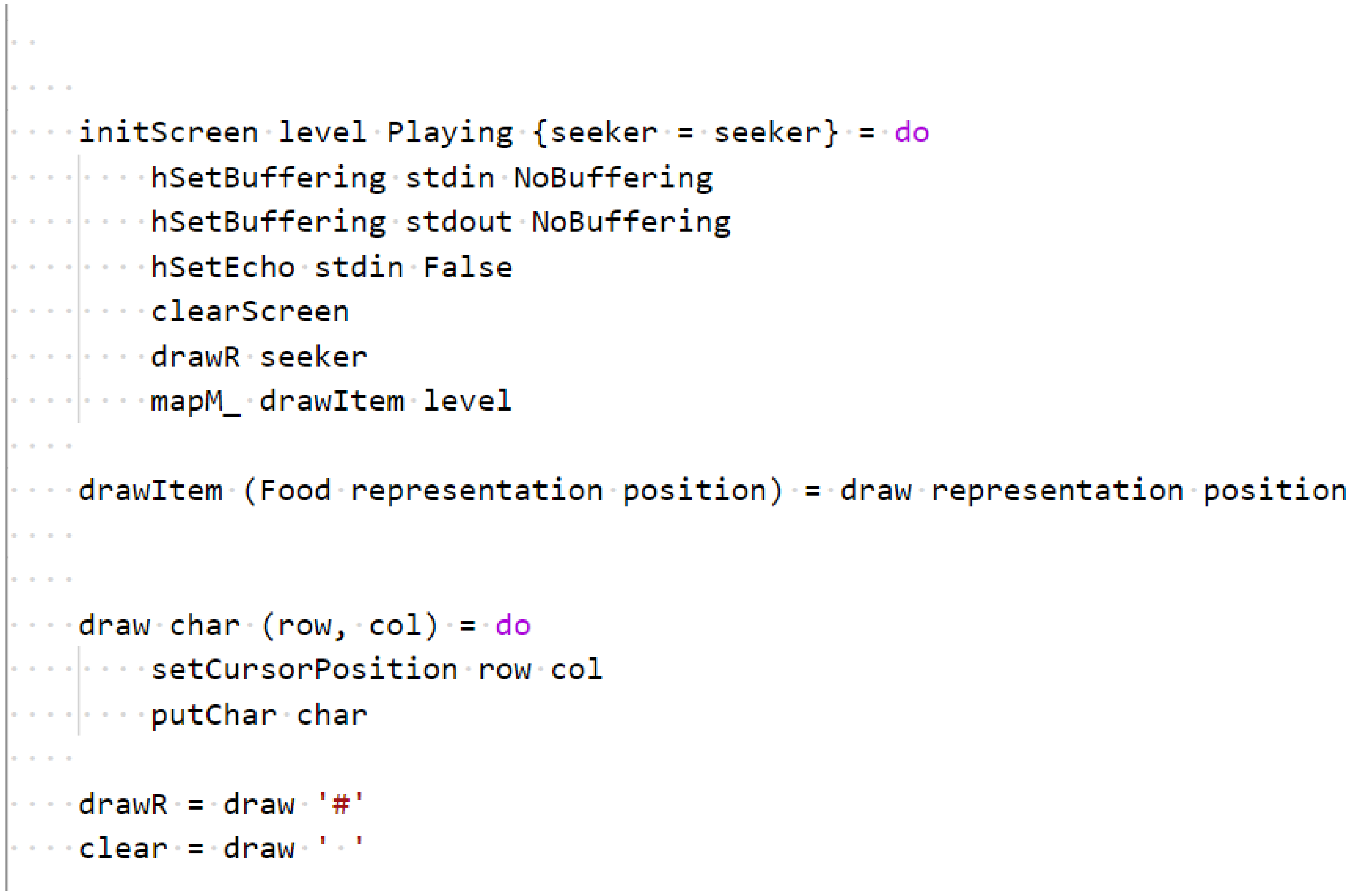
```haskell
data Command = MoveLeft
             | MoveDown
             | MoveUp
             | MoveRight
             | Quit
             | Unknown
             deriving (Eq)

parseInput :: [Char] -> [Command]
parseInput chars = map parseCommand chars

parseCommand :: Char -> Command
parseCommand 'q' = Quit
parseCommand 'h' = MoveLeft
parseCommand 'j' = MoveDown
parseCommand 'u' = MoveUp
parseCommand 'k' = MoveRight
parseCommand _ = Unknown
```



```haskell
  advance :: Level -> GameState -> Command -> GameState
  advance level state MoveLeft = moveSeeker level (0, -1) state
  advance level state MoveUp = moveSeeker level (-1, 0) state
  advance level state MoveDown = moveSeeker level (1, 0) state
  advance level state MoveRight = moveSeeker level (0, 1) state
  advance _ state Quit = state { message = "Goodbye!", over = True }
  advance _ state _ = state
```

- Step 3

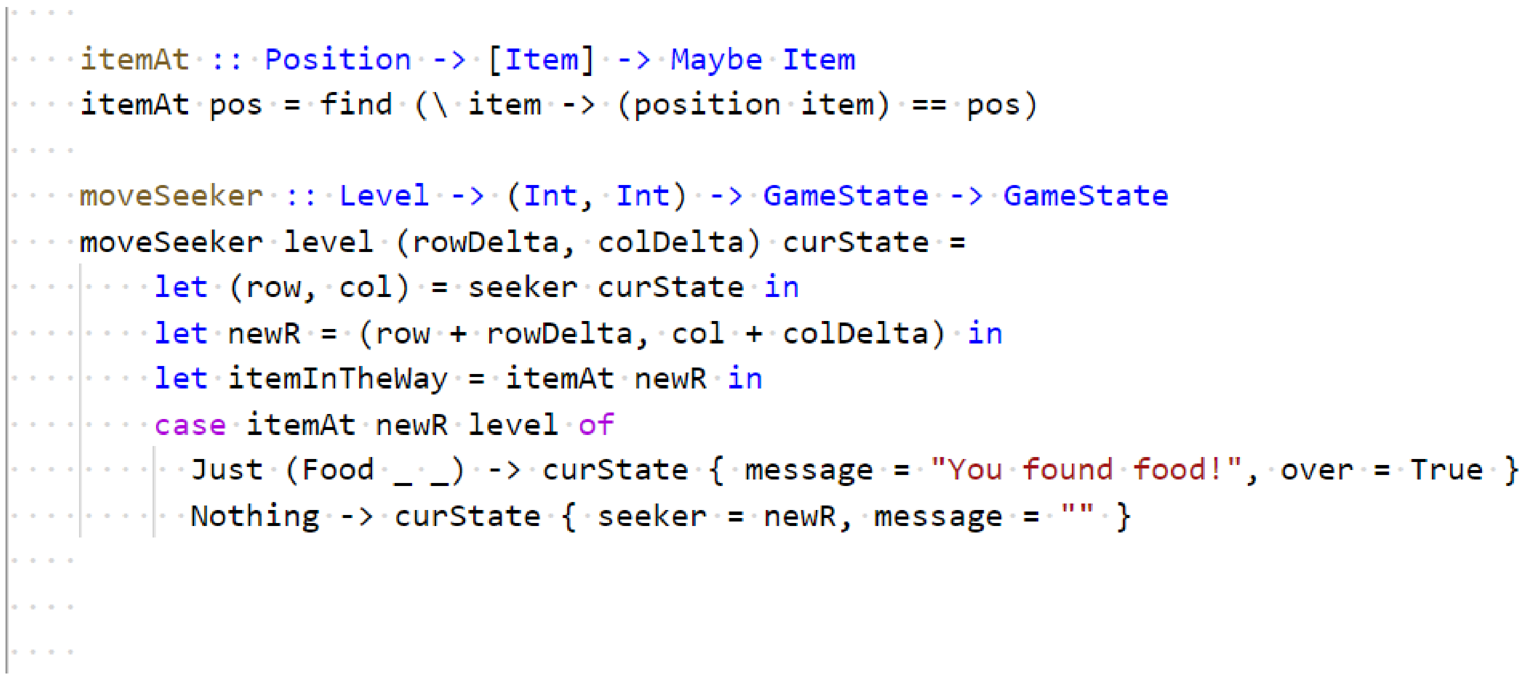
  Draw the seeker(“#”) and the food (**letters**) at the random position generated and initialize screen



```


    initScreen level Playing {seeker = seeker} = do
        hSetBuffering stdin NoBuffering
        hSetBuffering stdout NoBuffering
        hSetEcho stdin False
        clearScreen
        drawR seeker
        mapM_ drawItem level

    drawItem (Food representation position) = draw representation position



    draw char (row, col) = do
        setCursorPosition row col
        putChar char

    drawR = draw '#'
    clear = draw ' '
```

Step 4

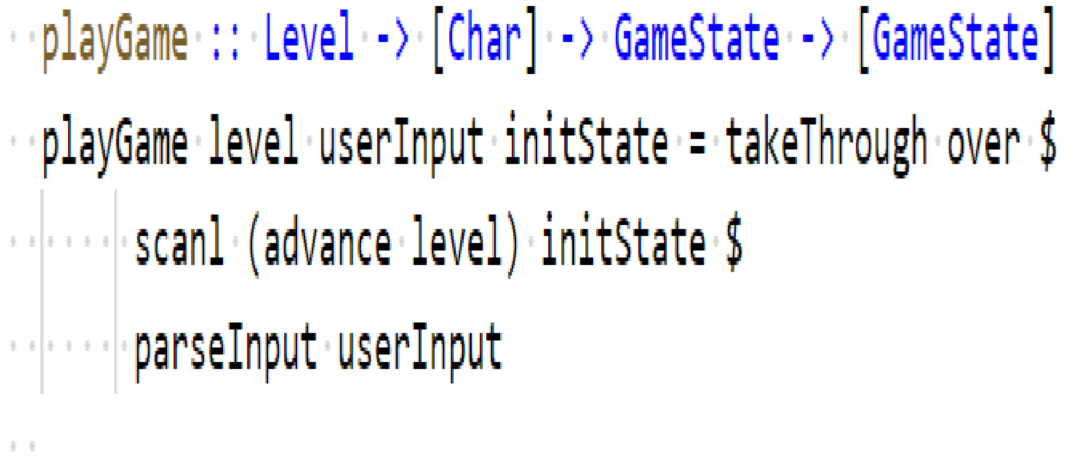
Check if the seeker found the food or not. If found then game over!



```
itemAt :: Position -> [Item] -> Maybe Item
itemAt pos = find (\ item -> (position item) == pos)

moveSeeker :: Level -> (Int, Int) -> GameState -> GameState
moveSeeker level (rowDelta, colDelta) curState =
    let (row, col) = seeker curState in
    let newR = (row + rowDelta, col + colDelta) in
    let itemInTheWay = itemAt newR in
    case itemAt newR level of
      Just (Food _ _) -> curState { message = "You found food!", over = True }
      Nothing -> curState { seeker = newR, message = "" }
```

- Step 5

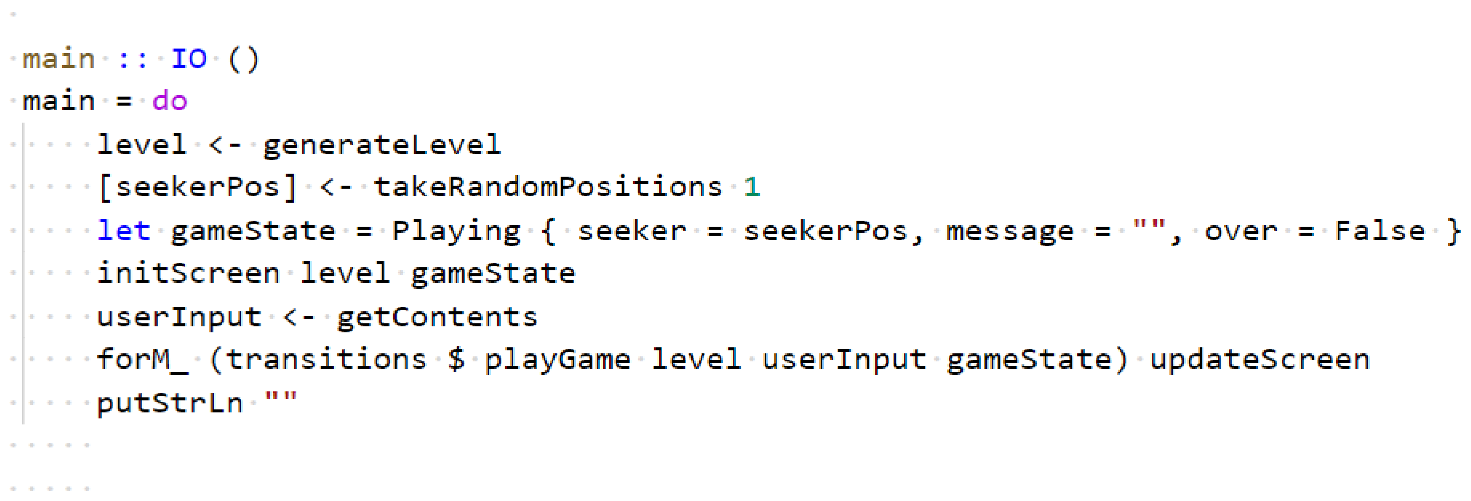
  Run the code till seeker finds the food



```
playGame :: Level -> [Char] -> GameState -> [GameState]
playGame level userInput initState = takeThrough over $
        scanl (advance level) initState $
        parseInput userInput
```

Step 6

Combine all these chunks of code and call the main function



```haskell
main :: IO ()
main = do
    level <- generateLevel
    [seekerPos] <- takeRandomPositions 1
    let gameState = Playing { seeker = seekerPos, message = "", over = False }
    initScreen level gameState
    userInput <- getContents
    forM_ (transitions $ playGame level userInput gameState) updateScreen
    putStrLn ""
```