МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования
«ИРКУТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ИГУ»)

Институт математики
и информационных технологий

Кафедра вычислительной
математики и оптимизации

ОТЧЕТ

по курсовой работе

Графический интерфейс с анимированной гирляндой в разных режимах. Переключение режимов автоматически и вручную. Перемещение лампочек.

Студента 3 курса группы 02321-ДБ
направления 01.03.02 «Прикладная
математика и информатика»
Булашкиновой Валентины Юрьевны

Научный руководитель:
К. т. н., доцент
Черкашин Евгений Александрович

Оценка
_____

Иркутск – 2022

Оглавление

1. Введение

Haskell — стандартизированный чистый функциональный язык программирования общего назначения. Является одним из самых распространённых языков программирования с поддержкой отложенных вычислений. Система типов — полная, сильная, статическая, с автоматическим выводом типов, основанная на системе типов Хиндли — Милнера. Поскольку язык функциональный, то основная управляющая структура — это функция.

Функциональное программирование — парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании последних (в отличие от функций как подпрограмм в процедурном программировании).

Цель данной курсовой работы — создать графический интерфейс с анимированной гирляндой в разных режимах на языке Haskell.

2. Теоретические основы

Графический интерфейс пользователя (ГИП), **графический пользовательский интерфейс (ГПИ)** — система средств для взаимодействия пользователя с электронными устройствами, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана (окон, значков, меню, кнопок, списков и т. п.).

Для курсовой работы будет использоваться **Gloss** – библиотека для работы с векторной графикой и использует OpenGL для работы с видеокартой.

3. Реализация

Опишем структуры данных на языке Haskell.

Перечисления «DiodColor» и «GarlandMode» отвечают за цвета диодов и режим работы гирлянды:

```
data DiodColor = RedColor | BlueColor | GreenColor deriving (Eq,Show,Bounded,
Enum)
```

Режим «RandomMode» настраивает гирлянду в соответствующим способом зависящий от координат диода. Режимы «RedMode» и «GreenMode» имеют один цвет, но отличаются способом зажигания и моргания диодов:

```
data GarlandMode = RandomMode | RedMode | GreenMode deriving
(Eq,Show,Bounded, Enum)
```

Структура «Diod» позволяет удобно хранить данные, а так же обращается к ним. Занимается хранением положения диода на экране пользователя, яркости, скорости изменения яркости, текущим цветом, состояния включен или выключен, и состояния движения диода. За счет манипуляции значениями «intensity» и «speedintensity» можно создавать различные анимации загорания и затухания, при правильной комбинации можно создавать анимацию движения огоньков:

```
data Diod = Diod{
   position :: (Float ,Float)
  ,intensity :: Float
  ,speedintensity :: Float
  ,dcolor :: DiodColor
  ,enable::Bool
  ,move::Bool
}deriving(Show)
```

Структура «Garland» хранит список диодов, семя для случайного значения, состояния работы, текущий режим работы и время работы, для смены режимов:

```
data Garland = Garland{
   gen :: StdGen
  ,diods :: [Diod]
  ,action :: Bool
  ,mode :: GarlandMode
  ,time::Float
}deriving(Show)
```

Функция «getColor» позволяет менять цвет диода на картинке и так же менять яркость. Яркость изменяется за счет прозрачности обрисовываемого объекта и уменьшением радиуса окружности:

```
getColor::Diod -> Picture -> Picture
getColor diod figue
  | dcolor diod == RedColor = Color (makeColorI 255 0 0 (roundFloatInt (255 *
abs (intensity diod)))) figue
  | dcolor diod == BlueColor = Color (makeColorI 0 0 255 (roundFloatInt (255
* abs (intensity diod)))) figue
  | dcolor diod == GreenColor = Color (makeColorI 0 255 0 (roundFloatInt (255
* abs (intensity diod)))) figue
```

Функция «diodToPicture» отрисовывает только один диод, меняет его положение и определяет цвет. А функция «diodsToPictures» уже собирает все изображения пропущенные через предыдущую функцию:

```
diodToPicture::Diod -> Picture
diodToPicture diod =
  if enable diod
    then
      getTranslate (position diod) (getColor diod $ circleSolid (radius * abs
(intensity diod)))
    else
      getTranslate (position diod) $ Color black $ circleSolid (radius * abs
(intensity diod))

diodsToPictures::[Diod] -> [Picture]
diodsToPictures [] = []
diodsToPictures (h:t) =
  if null t
    then [diodToPicture h]
    else  diodToPicture h : diodsToPictures t
```

Функция «getWires» занимается созданием проводов между диодами. Принимает на вход список диодов и по координатам определяет какие точки необходимо соединить. Функция «getWire» уже работает с двумя диодами, возвращая изображение линии:

```
getWires::[Diod]->[Picture]
getWires [] = []
getWires [h] = []
getWires (h1:h2:t) =
  if null t
    then
      [getWire h1 h2]
    else
      getWire h1 h2 : getWires (h2:t)
--получаем изображение проводов
getWire::Diod->Diod->Picture
getWire diod_1 diod_2 = Color white $ Line [position diod_1,position diod_2]
```

Функция «render» собирает все созданные изображения и формирует одно единое изображение, которое увидит пользователь приложения:

```
--собираем в общий список
render::Garland->Picture
render garland = pictures $ getWires (diods garland) ++ diodsToPictures
(diods garland)
```

Ниже представлено событие создания нового диода. При нажатии левой кнопки мыши срабатывает событие и запускается выполнение написанной нами функции. В переменных «xPas» и «yPas» передаются координаты мыши относительно окна приложения, которые становятся новыми координатам диода, так же заполняем все остальные поля:

```
handleEvents (EventKey (MouseButton LeftButton) Down _ (xPos, yPos)) garland
= garland {diods = ndiods }
  where
   ndiods = diods garland ++ [Diod (xPos, yPos) (cos (yPos/pi+ xPos))
0.015  (fst (random (gen garland))) True  False ]
```

Событие описанное ниже представляет собой захват диода. Алгоритм проходится по каждому диоду, смотрит первое попадание, и меняет флаг «move» на TRUE. И пока не произойдёт событие отпускания мыши, данное значение у данного диода не изменится.

```
handleEvents (EventKey (MouseButton RightButton) Down _ (xPos, yPos)) garland
= garland {diods = ndiods}
  where
   ndiods = getDiod (diods garland)
    where
      len::(Float,Float)->(Float,Float)->Float
      len (x,y) (x1,y1) = (x-x1)^2+(y-y1)^2
      getDiod [] = []
      getDiod (h:t) =
        if len (xPos,yPos) (position h) < radius^2
          then
            h {move = True} :  t
          else
            h : getDiod t
```

Событие ниже уже реализует отпускание взятых диодов:

```
handleEvents (EventKey (MouseButton RightButton) Up _ (xPos, yPos)) garland =
garland {diods = ndiods}
  where
  ndiods = getDiod (diods garland)
    where
      getDiod [] = []
      getDiod (h:t) =
            h {move = False} : getDiod t
```

Данное событие присваивает всем диодам, у которых «move» равен TRUE текущее значение координат мыши, создавая анимацию движения:

```
handleEvents (EventMotion (xPos, yPos)) garland = garland {diods = ndiods
(diods garland)}
  where
    ndiods [] = []
    ndiods (h:t) =
      if move h then
          h {position = (xPos, yPos)}: t
        else
          h : ndiods t
```

События меняют состояния диодов в зависимости от формул. Событие при нажатии на «1» закрашивает все диоды в красный цвет и включает моргание четных и нечетных диодов. При «2» каждый диод начинает постепенно моргать различными цветами. А «3» аналогичен первому режиму, все диоды светят зеленым цветов, но с постепенным медленным и ускоряющим движением от начало в конец, заканчивая вспышкой затухают и по заново:

```
handleEvents (EventKey (Char '1') Down _ _)
handleEvents (EventKey (Char '2') Down _ _)
handleEvents (EventKey (Char '3') Down _ _)
```

Основное место в программе занимает логика режимов. Функция «randomMode» реализует режим «2». Он изменяет цвета и интенсивность свечения. Для постепенного загорания и потухания значение «intensity». Для того чтобы оно было в переделах от -1 до 1, при превышении 1 оно меняет значение на -1 и снова возрастает. Во всех функция значение берется под модулем, тем самым оно меняется от 0 до 1 циклично не создавая резких переходов в анимации. Если это значение близко к 0, то диод меняет цвет в заданном порядке.

```
randomMode [] = []
    randomMode (h:t) = h {intensity = newintensity, dcolor =
newcolor}:randomMode t
      where
        newintensity = if intensity h > 1 then -1 else intensity h +
speedintensity h
        newcolor =
          if abs(intensity h) < 0.01
            then
              if dcolor h == RedColor
                then
```

```
                BlueColor
              else
                if dcolor h == BlueColor
                  then
                    GreenColor
                  else
                    RedColor
          else
            dcolor h
```

4. Интерфейс

Интерфейс представлен на рисунке 1 как черный экран. Данный цвет был выбран, чтобы было удобно видеть каждый диод. Для добавления лампочек необходимо нажать по окну левой кнопкой мыши в любое место.

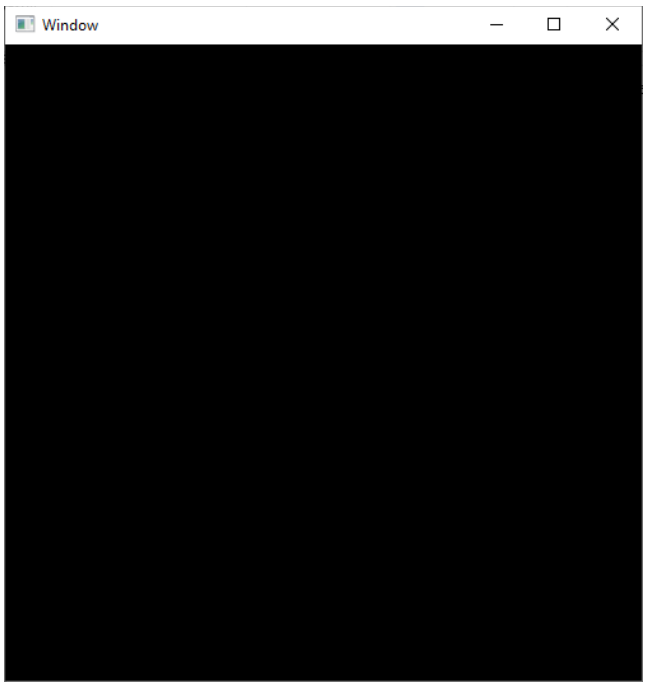

Рисунок 1 – Главный экран.

При множественном клике по экрану, можем получить следующий вид, представлен на рисунке 2.

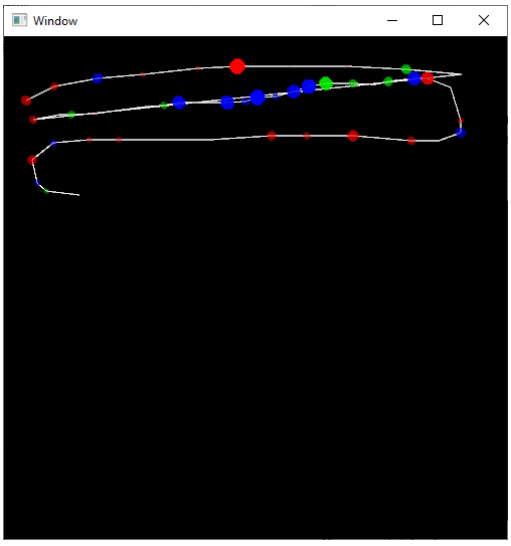

Рисунок 2 – Множество диодов.

Так же пользователю дается возможность передвигать светодиоды зажатием правой кнопки мыши. На рисунке 3 можно наблюдать изменения положения нижних диодов.

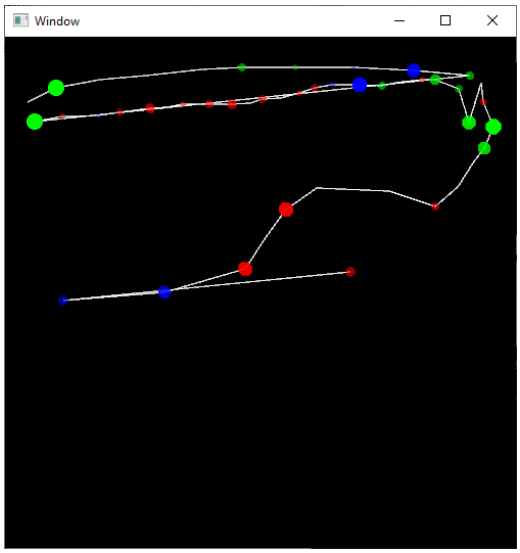

Рисунок 3 – Изменение положения диодов.

Пользователю так же доступны команды при нажатии на клавиши. В таблице 3 описаны клавиши и действия, связанные с ними.

Клавиша	Действие
1	Устанавливает всем диодам красный цвет, и вкл./выкл. четные и нечетные диоды каждые 2 секунды.
2	Устанавливает для каждого диода свой режим работы.
3	Устанавливает зеленый цвет, и загорание с задержкой.
0	Удаление всех диодов.
9	Вкл./выкл. автоматическую смену режимов.

5. Заключение

В процессе выполнения курсовой работы было изучено программирование в функциональном стиле, изучен язык программирования Haskell и система для создания и упаковки библиотек, и программ Haskell – Cabal.

Помимо языка была изучена библиотек Gloss, позволяющая в короткий срок быстро создавать графическое приложение (интерфейс).

6. Список литературы

1) Р. В. Душкин П. Практика работы на языке Haskell. М.:ДМК Пресс, 2010.-288с, ил. ISBN 978-5-94074-588-4
2) https://www.haskell.org/documentation/
3) https://hackage.haskell.org/package/gloss-1.13.2.1/docs/Graphics-Gloss.html

7. Приложение 1

```
module Main(main) where

import Graphics.Gloss
import Graphics.Gloss.Data.ViewPort
import Graphics.Gloss.Interface.Pure.Game
import GHC.Float.RealFracMethods (roundFloatInt)
import System.Random
import GHC.Num
import System.Posix.Internals (lstat)


--Настройка окна
width, height, offset :: Int
width = 500
height = 500
offset = 0


window :: Display
window = InWindow "Window" (width, height) (offset, offset)

background :: Color
background = black

radius::Float
radius = 8

--Структуры

data DiodColor = RedColor | BlueColor | GreenColor deriving (Eq,Show,Bounded,
Enum)
instance Random DiodColor where
    random g = case randomR (fromEnum (minBound :: DiodColor), fromEnum
(maxBound :: DiodColor)) g of
                  (r, g') -> (toEnum r, g')
    randomR (a,b) g = case randomR (fromEnum a, fromEnum b) g of
                      (r, g') -> (toEnum r, g')
data GarlandMode = RandomMode | RedMode | GreenMode deriving
(Eq,Show,Bounded, Enum)

data Diod = Diod{
   position :: (Float ,Float)
  ,intensity :: Float
  ,speedintensity :: Float
  ,dcolor :: DiodColor
  ,enable::Bool
  ,move::Bool
}deriving(Show)
```

```
data Garland = Garland{
   gen :: StdGen
  ,diods :: [Diod]
```

```
  ,action :: Bool
  ,mode :: GarlandMode
  ,time::Float
}deriving(Show)

--Отрисовка
getTranslate::(Float, Float)->Picture->Picture
getTranslate (x, y) = translate x y
getColor::Diod -> Picture -> Picture
getColor diod figue
  | dcolor diod == RedColor = Color (makeColorI 255 0 0 (roundFloatInt (255 *
abs (intensity diod)))) figue
  | dcolor diod == BlueColor = Color (makeColorI 0 0 255 (roundFloatInt (255
* abs (intensity diod)))) figue
  | dcolor diod == GreenColor = Color (makeColorI 0 255 0 (roundFloatInt (255
* abs (intensity diod)))) figue
--получаем изображение отдельного диода
diodToPicture::Diod -> Picture
diodToPicture diod =
  if enable diod
    then
      getTranslate (position diod) (getColor diod $ circleSolid (radius * abs
(intensity diod)))
    else
      getTranslate (position diod) $ Color black $ circleSolid (radius * abs
(intensity diod))

diodsToPictures::[Diod] -> [Picture]
diodsToPictures [] = []
diodsToPictures (h:t) =
  if null t
    then [diodToPicture h]
    else  diodToPicture h : diodsToPictures t

getWires::[Diod]->[Picture]
getWires [] = []
getWires [h] = []
getWires (h1:h2:t) =
  if null t
    then
      [getWire h1 h2]
    else
      getWire h1 h2 : getWires (h2:t)
--получаем изображение проводов
getWire::Diod->Diod->Picture
getWire diod_1 diod_2 = Color white $ Line [position diod_1,position diod_2]

--собираем в общий список
render::Garland->Picture
render garland = pictures $ getWires (diods garland) ++ diodsToPictures
(diods garland)


--События
handleEvents :: Event -> Garland -> Garland
```

```
handleEvents (EventKey (MouseButton LeftButton) Down _ (xPos, yPos)) garland
= garland {diods = ndiods }
  where
   ndiods = diods garland ++ [Diod (xPos, yPos) (cos (yPos/pi+ xPos)) 0.015
(fst (random (gen garland))) True  False ]

handleEvents (EventKey (MouseButton RightButton) Down _ (xPos, yPos)) garland
= garland {diods = ndiods}
  where
   ndiods = getDiod (diods garland)
    where
      len::(Float,Float)->(Float,Float)->Float
      len (x,y) (x1,y1) = (x-x1)^2+(y-y1)^2
      getDiod [] = []
      getDiod (h:t) =
        if len (xPos,yPos) (position h) < radius^2
          then
            h {move = True} :  t
          else
            h : getDiod t

handleEvents (EventKey (MouseButton RightButton) Up _ (xPos, yPos)) garland =
garland {diods = ndiods}
  where
  ndiods = getDiod (diods garland)
    where
      getDiod [] = []
      getDiod (h:t) =
            h {move = False} : getDiod t

handleEvents (EventMotion (xPos, yPos)) garland = garland {diods = ndiods
(diods garland)}
  where
    ndiods [] = []
    ndiods (h:t) =
      if move h then
          h {position = (xPos, yPos)}: t
        else
          h : ndiods t

handleEvents (EventKey (Char '0') Down _ _)  garland = garland {diods = []}

handleEvents (EventKey (Char '9') Down _ _)  garland = garland {action = not
$ action garland}

handleEvents (EventKey (Char '1') Down _ _)  garland = garland {mode =
RedMode, diods = ndiods (diods garland) 0}
  where
    ndiods::[Diod]->Integer->[Diod]
    ndiods [] i = []
    ndiods (h:t) i =
          h{speedintensity = fromIntegral (mod i 2) :: Float , intensity = -
1, dcolor = RedColor} : ndiods t (i+1)

handleEvents (EventKey (Char '2') Down _ _)  garland = garland {mode =
RandomMode, diods = ndiods (diods garland)}
```

```
  where
    ndiods::[Diod]->[Diod]
    ndiods [] = []
    ndiods (h:t)  =
          h{enable = True ,speedintensity = sni $ position h, intensity = ni
$ position h, dcolor = ncolor} : ndiods t
           where
             ni (x,y)= cos (y/pi + x)
             sni (x,y) = 0.015 + abs(sin(x/pi-y*0.2)*0.1)
             ncolor = fst $random $ gen garland

handleEvents (EventKey (Char '3') Down _ _)  garland = garland {mode =
GreenMode, diods = ndiods (diods garland) 0}
  where
    ndiods::[Diod]->Float->[Diod]
    ndiods [] i = []
    ndiods (h:t) i =
          h{enable = True, speedintensity = 0.015, intensity = -1 + 2/(i+1),
dcolor = GreenColor} : ndiods t (i+1)

handleEvents _ garland = garland
--Обновление


--
update :: Float -> Garland -> Garland
update tm garland = garland {diods = modeup (mode garland), time = ntime,
mode = nmode (mode garland)}
  where
    modeup::GarlandMode->[Diod]
    ntime = if not (action garland) && time garland > 10 then 0 else time
garland + tm
    nmode mod =
      if not (action garland) && time garland > 9 then  nnmode mod else mode
garland
    nnmode mode
      | mode == RedMode = RandomMode
      | mode == RandomMode = GreenMode
      | mode == GreenMode = RedMode
    modeup mode
      | mode == RedMode =redMode $ diods garland
      | mode == RandomMode = randomMode $ diods garland
      | mode == GreenMode = greenMode $ diods garland

    greenMode [] = []
    greenMode (h:t) = h {intensity = newintensity}:greenMode t
      where
        newintensity = if intensity h > 1 then -1 else intensity h +
speedintensity h

    redMode [] = []
    redMode (h:t)=
      h{enable = nenable, speedintensity = nsi, intensity = -1}: redMode t
        where
          nsi = if speedintensity h > 2 then 0 else speedintensity h + tm
          nenable = speedintensity h > 1
```

```
    randomMode [] = []
    randomMode (h:t) = h {intensity = newintensity, dcolor =
newcolor}:randomMode t
      where
        newintensity = if intensity h > 1 then -1 else intensity h +
speedintensity h
        newcolor =
          if abs(intensity h) < 0.01
            then
              if dcolor h == RedColor
                then
                  BlueColor
                else
                  if dcolor h == BlueColor
                    then
                      GreenColor
                    else
                      RedColor
            else
              dcolor h
main::IO()
main =
  do
    g <- newStdGen
    let garland = Garland g [] True RandomMode 0
    play window background fps garland render handleEvents update
  where
    fps = 30
```