复旦大學

函数式程序设计导引| Haskell课程设计

博士的家·增强版

任予琛 高正祺 刘诗玮

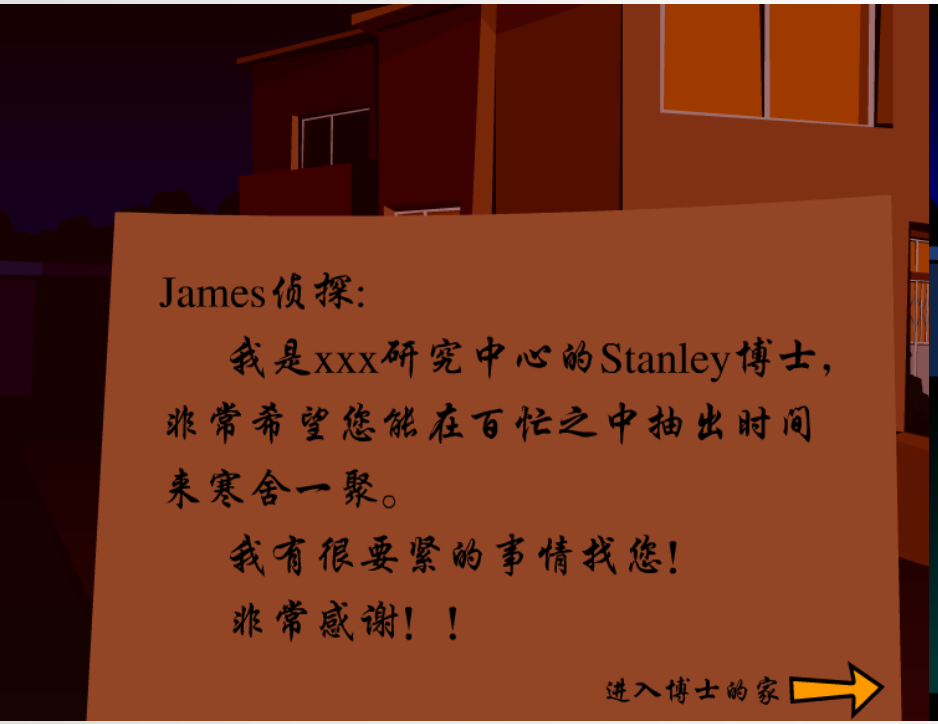
2018.12.28

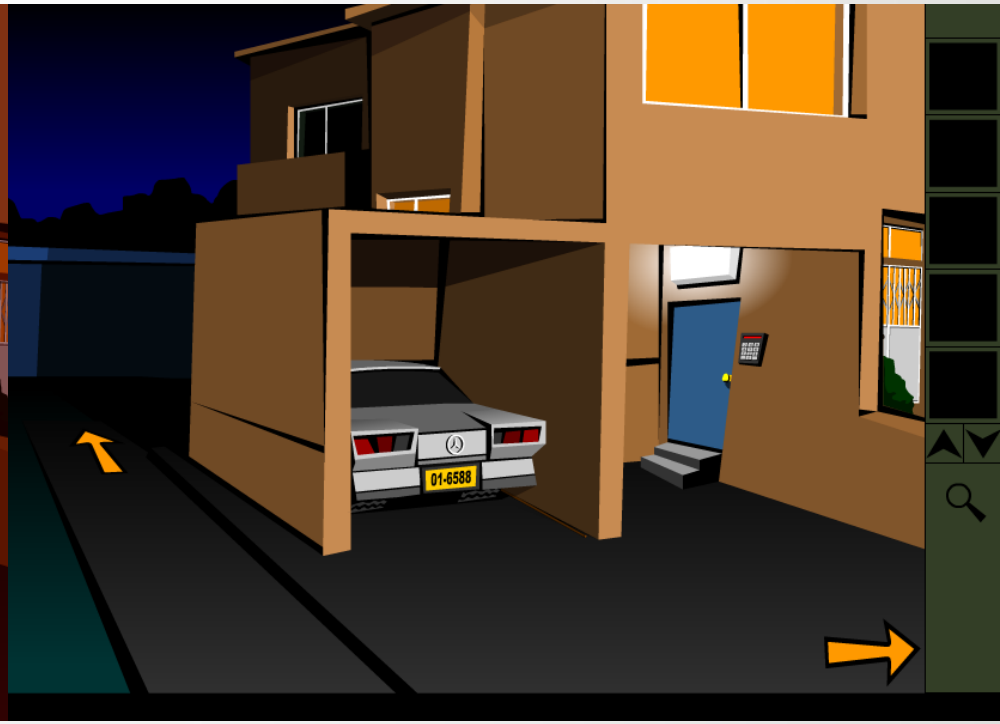
目录 Contents

C目录 ontents

Stanley 博士的家

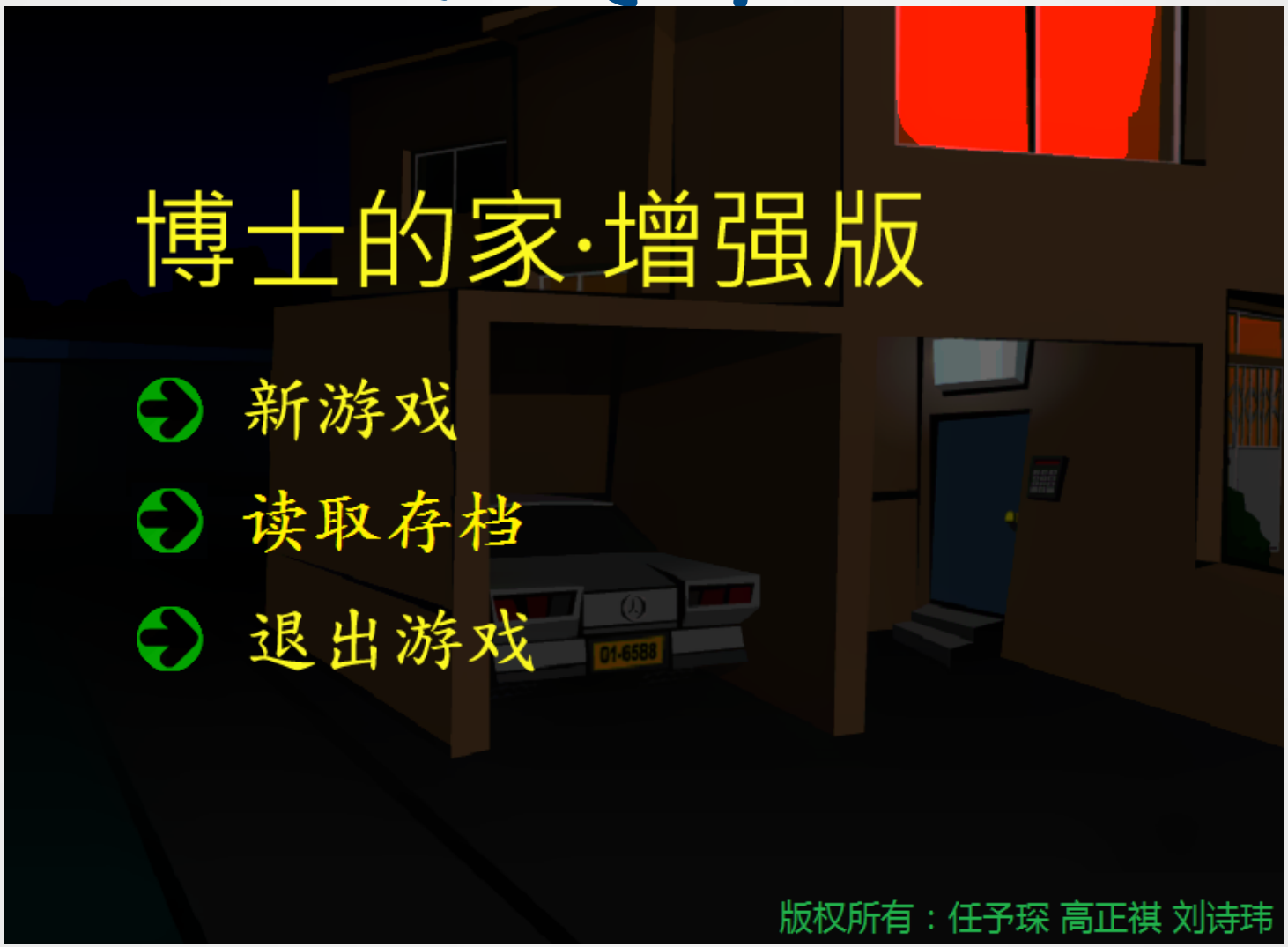
李鹏（James Li） 2005年





目录 Contents

游戏开始



游戏场景

物品栏

选中物品

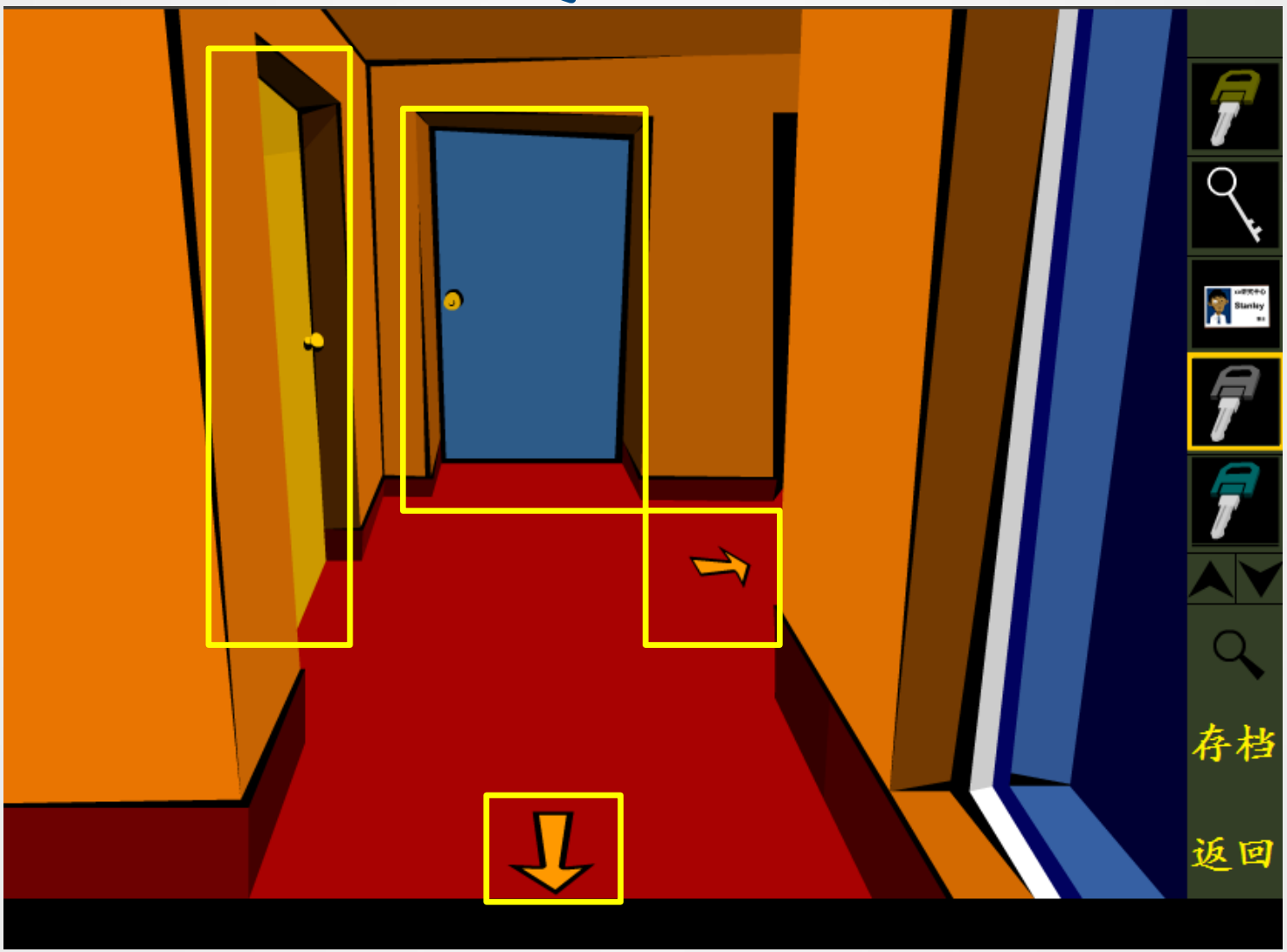
翻页

查看放大物品

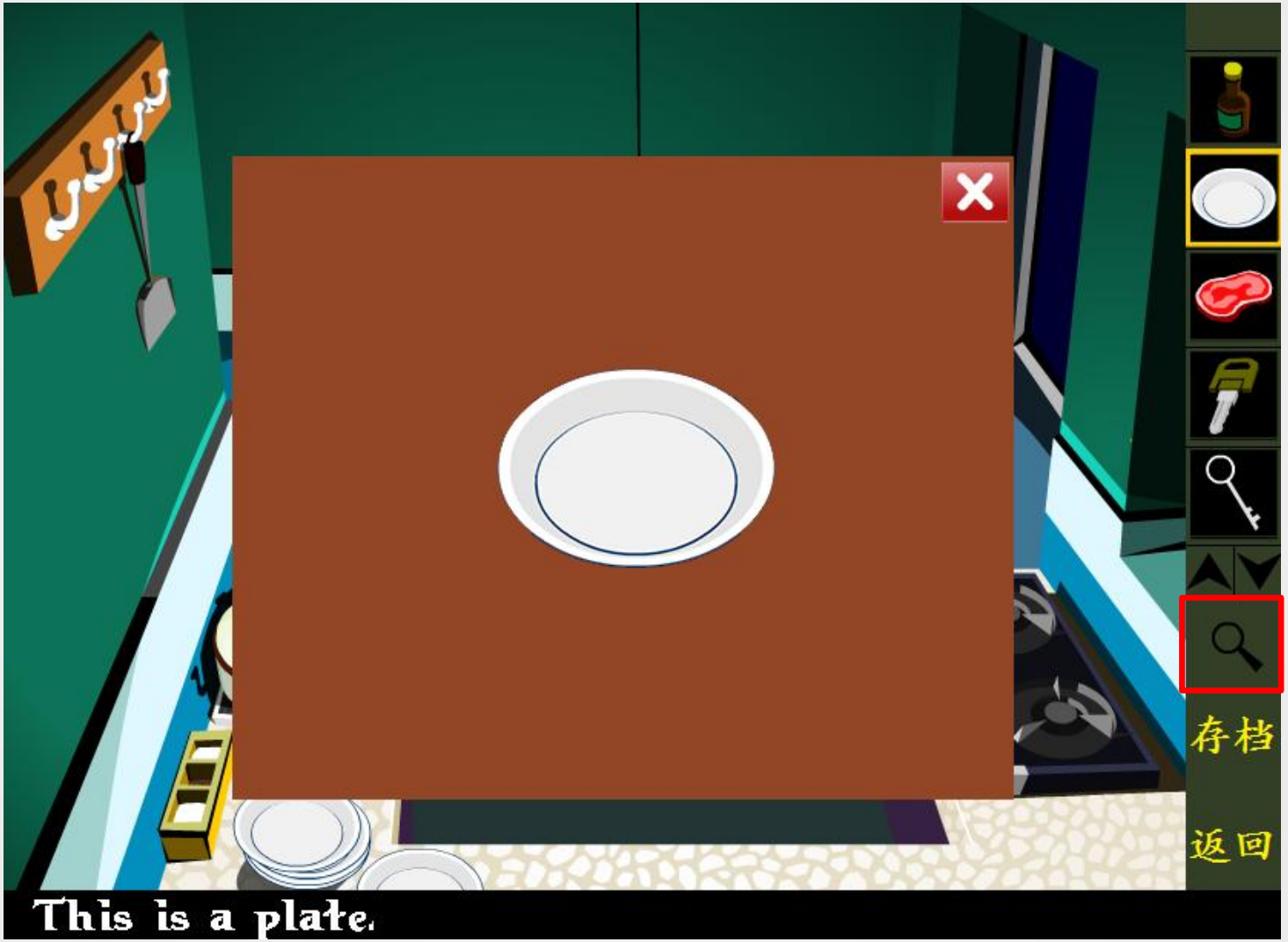
存档

返回

场景切换

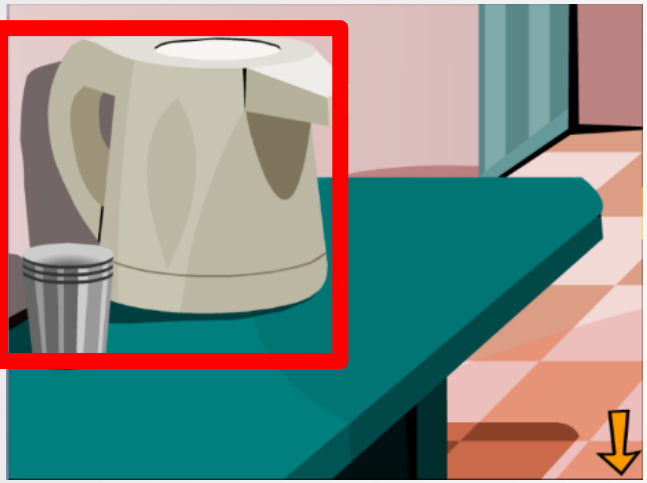
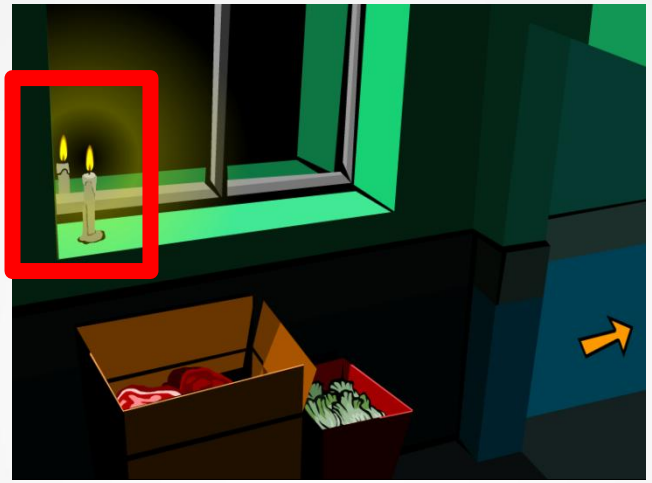
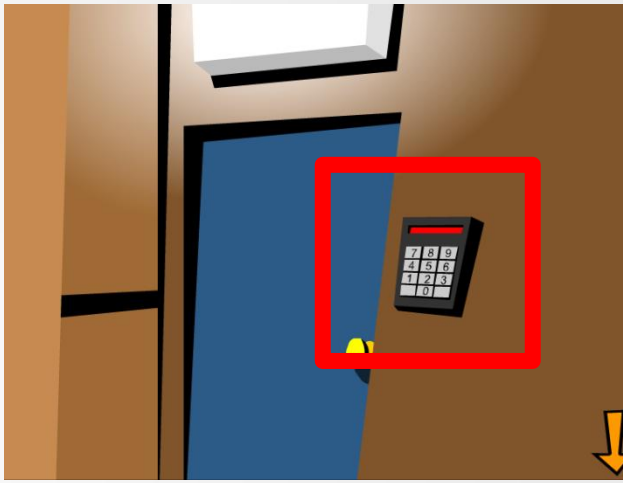
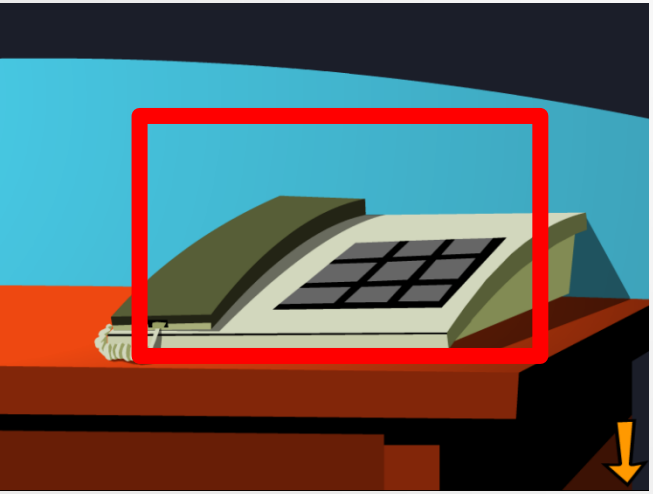
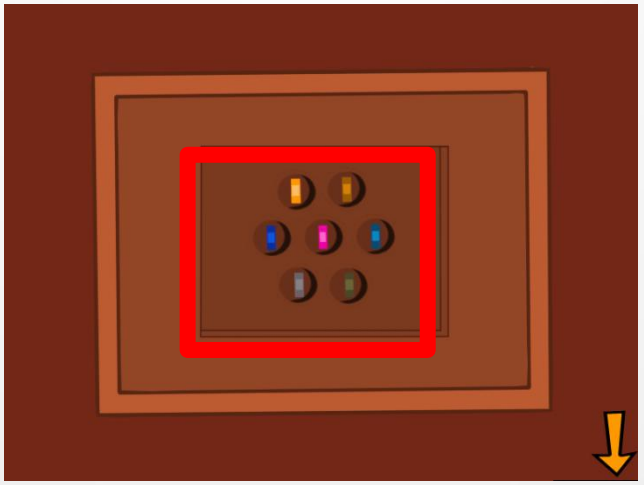


查看物品



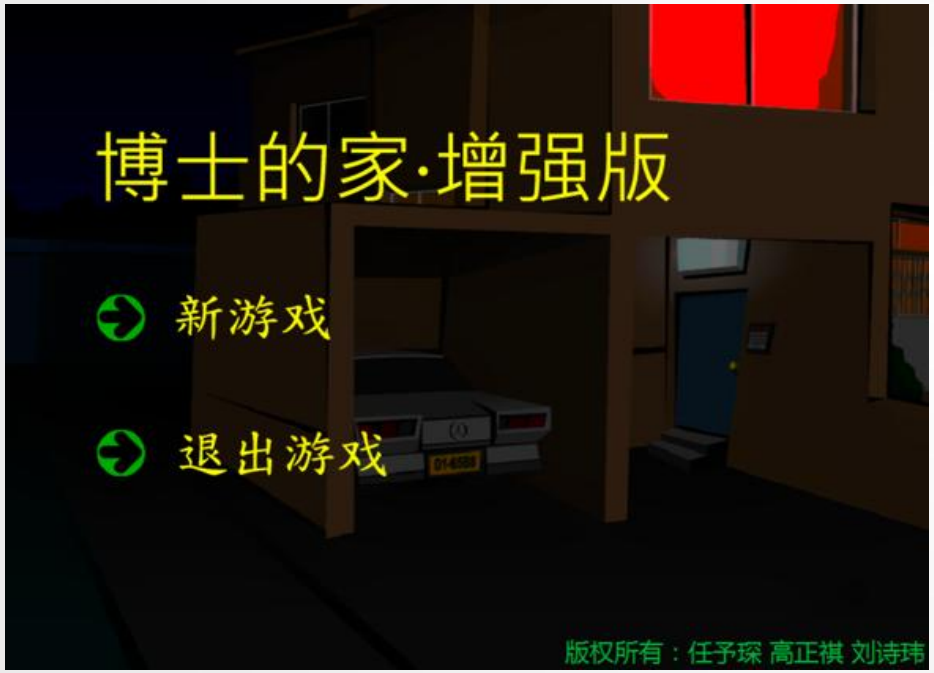
组合物品

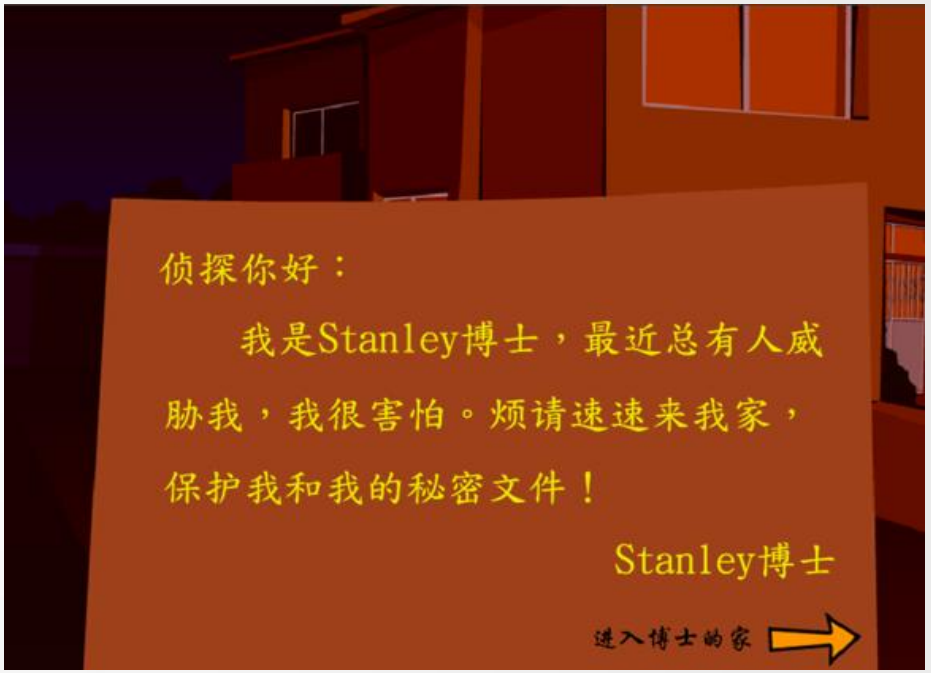
触发事件


目录 Contents

SDL2


博士的家·增强版
新游戏
退出游戏
版权所有：任予琛 高正祺 刘诗玮


侦探你好：
我是Stanley博士，最近总有人威胁我，我很害怕。烦请速速来我家，保护我和我的秘密文件！
Stanley博士
进入博士的家

SDL2

```
main = do
  SDL.initialize [SDL.InitVideo]
  window <- SDL.createWindow "SDL Tutorial" SDL.defaultWindow { SDL.windowInitialSize = V2 screenWidth screenHeight }
  SDL.showWindow window
  screenSurface <- SDL.getWindowSurface window

  xOut <- getDataFileName "figs/first.bmp" >>= SDL.loadBMP
  xOut2 <- getDataFileName "figs/second.bmp" >>= SDL.loadBMP
  SDL.surfaceBlit xOut Nothing screenSurface Nothing
  let
    loop mouse buttons = do
      --events  <- SDL.pollEvents
      events <- map SDL.eventPayload <$> ((<$>) (\a -> a:[]) SDL.waitEvent)
      mousePos <- SDL.getAbsoluteMouseLocation

      let current_state = getAny $ foldMap (\case
                SDL.MouseButtonEvent e -> Any $ SDL.mouseButtonEventMotion e == SDL.Pressed
                otherwise -> Any False) events

      let quit = SDL.QuitEvent `elem` events
      let mouse' = Mouse {posM = mousePos, state = current_state}
      let res = map (handleEvent mouse') buttons
      if (head res) == True
         then SDL.surfaceBlit xOut2 Nothing screenSurface Nothing
         else return $  Just (SDL.Rectangle (P (V2 0 0)) (V2 0 0))
      SDL.updateWindowSurface window
      unless (quit || (last res)) (loop mouse' buttons)

  loop (Mouse {posM = (P (V2 0 0)), state = False}) [button1,button2]

  SDL.freeSurface xOut
  SDL.destroyWindow window
```

SDL2

```haskell
import Paths_sdl2 (getDataFileName)
screenWidth, screenHeight :: CInt
(screenWidth, screenHeight) = (640, 480)
main :: IO ()
main = do
  SDL.initialize [SDL.InitVideo]
  window <- SDL.createWindow "SDL Tutorial" SDL.defaultWindow { SDL.windowInitialSize = V2 screenWidth screenHeight }
  SDL.showWindow window
  screenSurface <- SDL.getWindowSurface window

  helloWorld <- getDataFileName "examples/lazyfoo/hello_world.bmp" >>= SDL.loadBMP
```

```haskell
getDataFileName :: FilePath -> IO FilePath
getDataFileName = return
```

音乐播放

背景音乐支持库

- SDL2_MIXER

  SDL2_MIXER是SDL2在音频支持上的扩展

音效与背景音乐

- SDL2_MIXER将音频分为音效和背景音乐

  音效(sound)：可以同时播放多个音效

  音乐(music)：只能有一个背景音乐播放

背景音乐播放流程

```
SDL.initialize [SDL.InitAudio]                                  → 初始SDL_MIXER
Mix.openAudio 44100 Mix.AUDIO_S16LSB 2 4096                     → 打开播放设备
Mix.allocateChannels 16                                         → 设置混音通道(mix channel)数
music <- withCString "./bgm.mp3" $ \cstr -> Mix.loadMUS cstr    → 加载音频文件
temp <- Mix.playMusic music (-1)                                → 播放音频
assert $ temp /= -1                                             → 是否正常播放
threadDelay 100000                                              → 等待音频播放完成
```

- 关键函数

  openAudio

  allocateChannels

  threadDelay

音乐播放

音效播放流程

```
SDL.initialize [SDL.InitAudio]
Mix.openAudio 44100 Mix.AUDIO_S16LSB 2 4096
Mix.allocateChannels 16
sound <- withCString "./bgm.wav" $ \cstr -> Mix.loadWav cstr
channel <- Mix.playChannel (-1) sound 0
assert $ channel /= -1
threadDelay 100000
```

初始SDL_MIXER

打开播放设备

设置混音通道(mix channel)数

加载音频文件

播放音频

是否正常播放

等待音频播放完成

- 关键函数
  - loadWav
  - playChannel

文本显示

文本显示支持库

- SDL2_TTF

  SDL2不支持文本显示

文本显示流程

```
white :: SDL.Font.Color
white = SDL.V4 255 255 255 0

Font.initialize
font <- Font.load "films.Hellbound.ttf" 30
text <- Font.solid font red "There is the dog"
Font.free font
SDL.surfaceBlit text Nothing screenSurface (Just (P (V2 20 575)))
```

字体颜色

初始TTF

加载字库，设置字号

加载文本

释放字库

文本显示

- 通过SDL2_MIXER，文本作为图片显示

Haskell调用其他语言函数

- FFI (Foreign Function Interface)

  通过FFI，Haskell可以调用其他语言的函数，也可以被其他语言调用

- FFI 使用流程

  文件首行添加 {-# LANGUAGE ForeignFunctionInterface #-}

  将其他语言函数原型转换为Haskell中相对应的原型

```
foreign import ccall "SDL_mixer.h Mix_LoadMUS" loadMus' :: CString -> IO (Ptr Music)
```

- 数据类型转换

  外部语言的数据类型，Haskell往往不支持

模块	类型
Foreign.C.Types	CInt，CDouble，CUChar...
Foreign.C.String	CString
Foreign.C.Ptr	指针

Haskell调用其他语言函数

- Haskell可以调用外部语言的原因

  在机械码层面上实现语言的转换

- 实现自己的SDL2_TTF库

  根据FFI，实现了SDL2_TTF库基本函数

```
foreign import ccall "TTF_SizeText" sizeText' :: Ptr Font -> CString -> Ptr CInt -> Ptr CInt -> IO CInt
foreign import ccall "TTF_SizeUTF8" sizeUTF8' :: Ptr Font -> CString -> Ptr CInt -> Ptr CInt -> IO CInt
foreign import ccall "TTF_SizeUNICODE" sizeUNICODE' :: Ptr Font -> Ptr CUShort -> Ptr CInt -> Ptr CInt -> IO CInt

init :: MonadIO m => m CInt
init = liftIO init'
{-# INLINE init #-}

quit :: MonadIO m => m ()
quit = liftIO quit'
{-# INLINE quit #-}

wasinit :: MonadIO m => m CInt
wasinit = liftIO wasinit'
{-# INLINE wasinit #-}

load :: MonadIO m => CString -> CInt -> m (Ptr Font)
load v1 v2 = liftIO $ load' v1 v2
{-# INLINE load #-}
```

游戏素材获取与整理

- 游戏界面各元素排列组合可能极多
- 获取游戏素材是一项工作量巨大又无趣的工作

批处理脚本

函数名	函数功能
screenshot	鼠标控制截图区域并保存图片；打印图片左上角坐标并保存
batch_rename	某些图片漏截后，重新修改文件夹下所有图片名称，保证截取顺序不变
batch_resize	统一手动截取的图片大小

- 利用OpenCV批处理图片减少工作量
- 仍需要大量手动工作截取原始游戏画面，初始化各数据结构

游戏演示


Doctor's House - Enhanced Version
博士的家·增强版
新游戏
读取存档
退出游戏
版权所有：任予琛 高正祺 刘诗玮

目录 Contents

main :: IO ()

（1）开启音乐

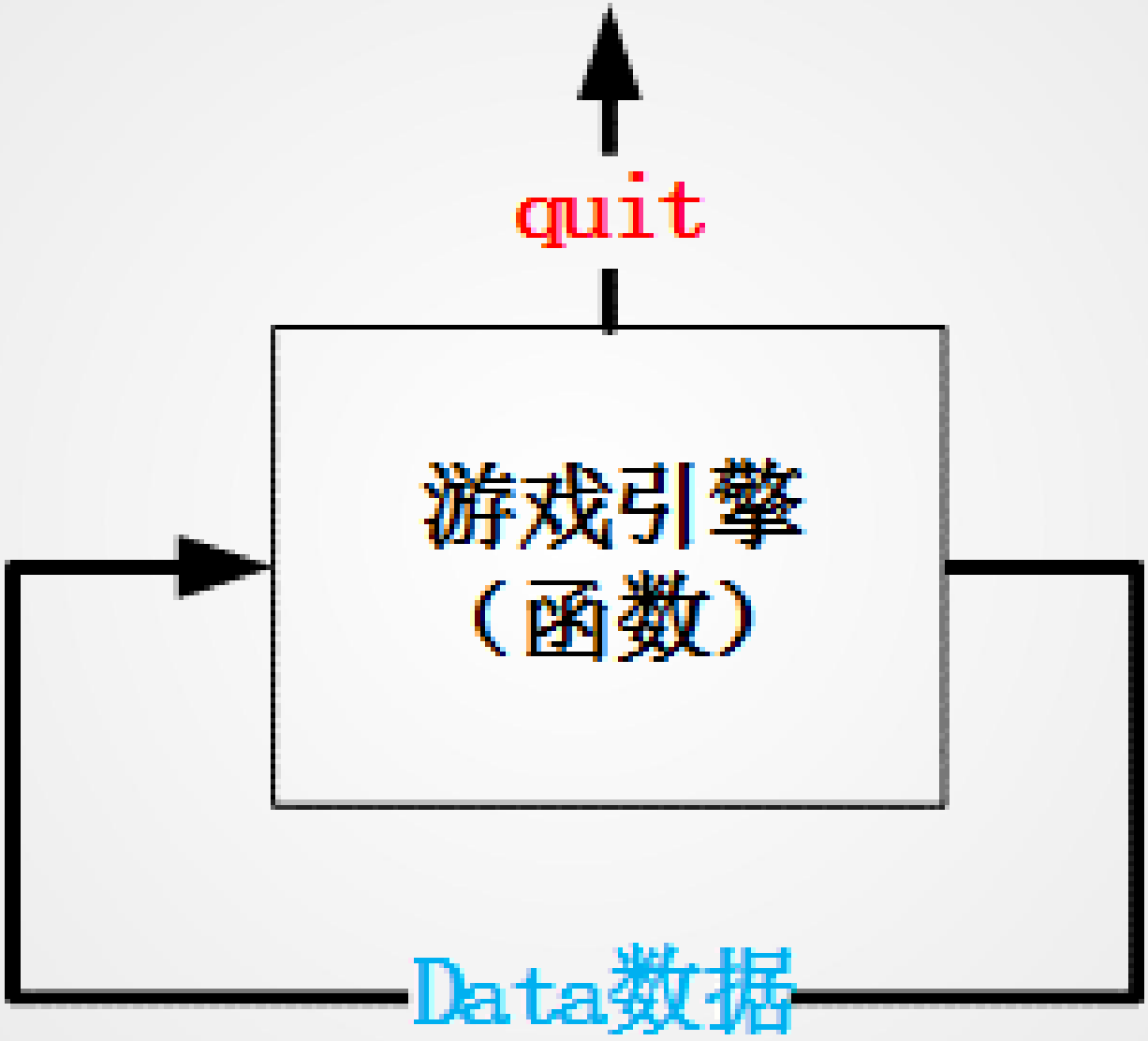
（2）设置文本字体

（3）创建窗口

（4）绘制游戏开始界面

（5）进入游戏引擎

（6）关闭窗口


quit
游戏引擎
（函数）
Data数据

```
data Interface = Interface {  dialogue           :: Int
                            , possessed_items    :: [Int]
                            , selected_item      :: Int
                            , showed_first_item  :: Int
                            , main_scene         :: Int
                            , current_scene      :: Int
                           }
```

当前场景、物品、对话……

```
data Scene = Scene {  background :: Int
                   , triggers   :: [Int]
                   }
```

背景图、触发域列表

动态游戏资源 · 触发域 Trigger

```
data Trigger = Trigger {  picture :: Int
                       , trigger_vertex :: (Point V2 CInt)
                       , trigger_size :: V2 CInt
                       , trigger_type :: Triggertype
                       , needed_item :: Int
                       , backup_item :: Int
                       , trigger_data :: Int
                       , relation_trigger :: Int
                       , relation_trigger_new_scene :: Int
                       , relation_trigger_new_picture :: Int
                       , triggerself_new_picture :: Int
                       , fail_dialogue :: Int
                       , new_dialogue :: Int
                       , new_scene :: Int
                       , add_item :: Int
                       , delete_item :: Int
                       , selected_item_delete :: Bool
                       , change_main_scene :: Bool
                       , relation_scene :: Int
                       , relation_scene_trigger_delete :: Int
                       , relation_scene_trigger_add :: Int
                       , triggerself_delete :: Bool
                       }  deriving (Eq)
```

游戏资源 · 物品、密码锁、钥匙插盘……

```
data Item = Item {  item_picture :: Int
                  , item_scene :: Int
                  , trigger_picture :: Int
                  }
```

```
data Password_Lock = Password_Lock { correct_password :: [Int]
                                   , input_password   :: [Int]
                                   }
```

```
data Advanced_Password_Lock = Advanced_Password_Lock { ad_correct_password :: [Int]
                                                     , ad_hidden_password  :: [Int]
                                                     , ad_input_password   :: [Int]
                                                     }
```

```
data Keys_Lock = Keys_Lock { correct_keys :: [Int]
                           , current_keys :: [Int]
                           }
```

绘制游戏界面：draw_interface

```
let
  draw_interface interface all_scenes all_triggers = do
```

（1）边栏底图

（2）物品选框，物品

（3）对话提示

（4）主场景及触发域

（5）当前场景及触发域

游戏引擎：loop_run

```
let
  loop_run interface all_scenes all_triggers house_password_lock bronze_password_lock file_password_lock rainbow_lock = do
    event <- map SDL.eventPayload <$> ((<$>) (\a -> a:[]) SDL.waitEvent)  --get [one event]
    mousePos <- SDL.getAbsoluteMouseLocation  --get mouse position

    let whether_click = getAny $ foldMap (\case
            SDL.MouseButtonEvent e -> Any $ SDL.mouseButtonEventMotion e == SDL.Pressed
            otherwise -> Any False) event
    let quit = SDL.QuitEvent `elem` event

    if whether_click
      then  --deal with click event
        if (within_ra_pos mousePos ra_magnifier) && (selected_item interface >= 0)  --click magnifier
        then do let new_interface = event_magnifier interface
                draw_interface new_interface all_scenes all_triggers
                unless (quit) (loop_run new_interface all_scenes all_triggers house_password_lock bronze_password_lock fil
        else if (within_ra_pos mousePos ra_turn_page_up) && (showed_first_item interface > 0)  --click turn page up
             then do let new_interface = event_turnpageup interface
                     draw_interface new_interface all_scenes all_triggers
                     unless (quit) (loop_run new_interface all_scenes all_triggers house_password_lock bronze_password_loc
        else if (within_ra_pos mousePos ra_record) && (not (elem (main_scene interface) [87, 88, 89, 90]))
             then do r1 <- record_interface interface
                     r3 <- record_all_triggers all_triggers
                     r4 <- record_all_locks house_password_lock bronze_password_lock file_password_lock rainbow_lock
```

触发域、物品栏、读/存档、返回、退出

尝试触发域：try_triggers

```
try_triggers :: (Point V2 CInt) -> Interface -> [
try_triggers mousePos interface all_scenes all_tr
  | (my_trigger == []) && (fail_trigger == []) =
  | fail_trigger /= [] = (False, (update_dialogue
  | otherwise = trigger_event my_trigger_index in
```

（1）非触发域

（2）触发失败

（3）触发成功

触发事件/更新资源：trigger_event

```
trigger_event :: Int -> Interface -> [Scene] -> [Trigger] -> Password_Lock -> Password_Lock ->
trigger_event my_tri_index interface all_scenes all_triggers house_password_lock bronze_passwo
  | (trigger_type my_tri == HOUSE_PASSWORD_BUTTON) && (judge_open house_password_lock) = (Fals
  | ((trigger_type my_tri) == HOUSE_PASSWORD_BUTTON) && (not (judge_open house_password_lock))
  | (trigger_type my_tri) == BRONZE_PASSWORD_BUTTON = (False, bronze_new_interface, all_scenes
  | (trigger_type my_tri) == ADVANCED_PASSWORD_BUTTON = (False, advanced_new_interface, all_sc
  | (trigger_type my_tri) == KEY_HOLE = (False, rainbow_new_interface, all_scenes, rainbow_new
  | (trigger_type my_tri) == MENU = (False, init_interface, init_scenes, init_triggers, house_
  | (trigger_type my_tri) == END_GAME = (True, interface, all_scenes, all_triggers, house_pass
  | otherwise = (False, ordinary_new_interface, ordinary_new_all_scenes, ordinary_new_all_trig
      where my_tri = all_triggers !! my_tri_index
```

密码锁按钮、钥匙孔、返回、退出、普通

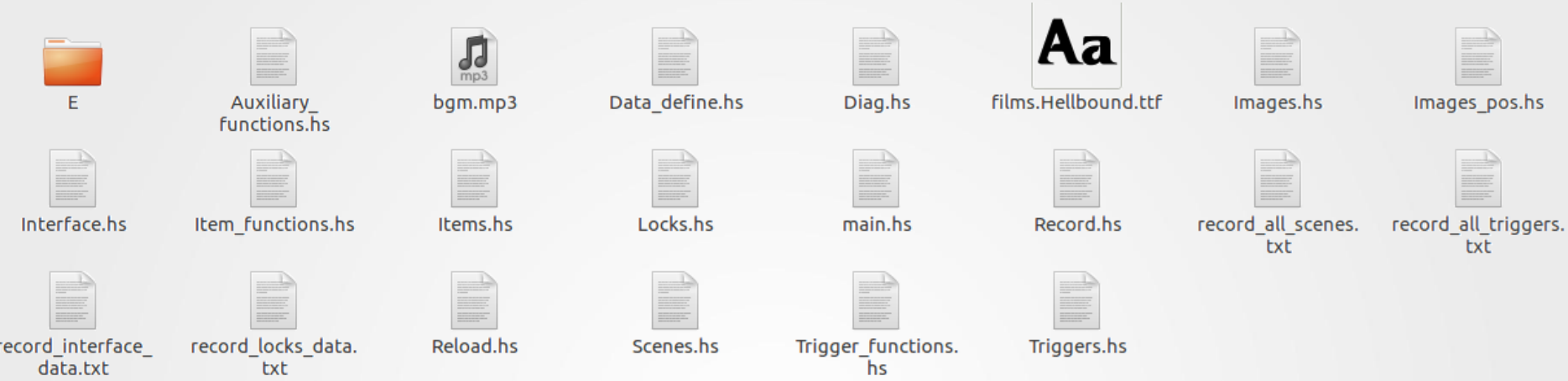
点击物品栏：event_

```haskell
event_magnifier :: Interface -> Interface
event_magnifier interface = if elem (main_scene interface) [87, 88, 89, 90]
                            then interface
                            else interface { dialogue = ((possessed_items ir
                                                        , current_scene = item_scene $ al


--click turn_page_up
event_turnpageup :: Interface -> Interface
event_turnpageup interface = interface { showed_first_item = (showed_first_i


--click turn_page_down
event_turnpagedown :: Interface -> Interface
event_turnpagedown interface = interface { showed_first_item = (showed_first


--click item_1
event_item1 :: Interface -> Interface
event_item1 interface = interface { selected_item = (showed_first_item inter
```

存/读档：_record/reload_

```
record_all_scenes :: [Scene] -> IO (Bool)
record_all_scenes all_scenes = do handle <- openFile "record_all_scenes.txt" WriteMode
                                  h_is_open <- hIsOpen handle
                                  if h_is_open
                                  then do loop_record_all_scenes handle all_scenes
                                          hClose handle
                                          return (True)
                                  else return (False)

loop_record_all_scenes :: Handle -> [Scene] -> IO ()
loop_record_all_scenes handle all_scenes = do if length all_scenes > 0
                                              then do let cur_scene = head all_scenes
                                                      hPutStrLn handle (show (background cur_scene))
                                                      hPutStrLn handle (show (triggers cur_scene))
                                                      loop_record_all_scenes handle (tail all_scenes)
                                              else return ()
reload_interface :: Interface -> IO Interface
reload_interface interface = do handle <- openFile "record_interface_data.txt" ReadMode
                                h_is_open <- hIsOpen handle
                                if h_is_open
                                then do content_dialogue <- hGetLine handle
                                        content_possessed_items <- hGetLine handle
                                        content_selected_item <- hGetLine handle
                                        content_showed_first_item <- hGetLine handle
                                        content_main_scene <- hGetLine handle
                                        hClose handle
                                        return Interface { dialogue = str2int content_dialogue
                                                         , possessed_items = str2intlist content_possessed_items
                                                         , selected_item = str2int content_selected_item
                                                         , showed_first_item = str2int content_showed_first_item
                                                         , main_scene = str2int content_main_scene
                                                         , current_scene = str2int content_main_scene }
                                else return interface
```

整理、编译、链接


E
Auxiliary_ functions.hs
bgm.mp3
Data_define.hs
Diag.hs
films.Hellbound.ttf
Images.hs
Images_pos.hs
Interface.hs
Item_functions.hs
Items.hs
Locks.hs
main.hs
Record.hs
record_all_scenes. txt
record_all_triggers. txt
record_interface_ data.txt
record_locks_data. txt
Reload.hs
Scenes.hs
Trigger_functions. hs
Triggers.hs


E
bgm.mp3
films.Hellbound.ttf
main
record_all_scenes. txt
record_locks_data. txt
record_all_triggers. txt
record_interface_ data.txt

目录 Contents

成果

（1）具有图形用户界面、动态文本、背景音乐、原创Haskell代码的解密游戏，以及配套的用户手册。

（2）丰富有趣的游戏功能：多类型的触发事件、组合物品、密码锁、钥匙插盘、读/存档……

展望

（1）可读性更高、更便捷的游戏资源编辑器

（2）动画、多份存档等高级功能

（3）全新的游戏场景和剧情

Haskell特色代码 · 举例

```haskell
cur_triggers = [all_triggers !! index | index <- (triggers cur_scene)]

cur_triggers_images = filter (>= 0) (map picture cur_triggers)

change_list_x ls n new = (take n ls) ++ (new : (drop (n+1) ls))

filt_core_str = map (\ch -> if ch == ',' then ' ' else ch) core_str
```

```haskell
char2int ch
  |ch == '1' = 1
  |ch == '2' = 2
  |ch == '3' = 3
  |ch == '4' = 4
  |ch == '5' = 5
  |ch == '6' = 6
  |ch == '7' = 7
  |ch == '8' = 8
  |ch == '9' = 9
  |otherwise = 0
```

11月初 ~ 12月底，每周讨论

任予琛：游戏逻辑代码……

高正祺：SDL2图形库……

刘诗玮：音乐文字库……

详见《报告》中的日程表

参考资料（库）

[1] sdl2[OL]. *http://hackage.haskell.org/package/sdl2*.

[2] Haskell SDL2 Examples[OL]. *https://github.com/palf/haskell-sdl2-examples*.

[3] sdl2-ttf[OL]. *https://github.com/haskell-game/sdl2-ttf*.

[4] sdl2-mix[OL]. *https://github.com/tempname11/sdl2-mixer*.

復旦大學

Thanks!

任予琛 高正祺 刘诗玮

2018.12.28