SDD Major work

Logic puzzler | HOME

Rory Symonds

[Year]

Contents

[Algorithms 2](#_Toc517366934)

[Rationale: 2](#_Toc517366935)

[Algorithms 2](#_Toc517366936)

[ScreenMsg 3](#_Toc517366937)

[Rectbutton 4](#_Toc517366938)

[Imgbutton 5](#_Toc517366939)

[SelectLevel 6](#_Toc517366940)

[Gamescreen 7](#_Toc517366941)

[Main 11](#_Toc517366942)

[Mainline Algorithm 14](#_Toc517366943)

[DATA DICTIONARY 15](#_Toc517366944)

[Rationale: 15](#_Toc517366945)

[Data Dictionary 15](#_Toc517366946)

[Source Code 18](#_Toc517366947)

[Rationale: 18](#_Toc517366948)

[Source Code 18](#_Toc517366949)

[Structure Diagram 26](#_Toc517366950)

[Rationale: 26](#_Toc517366951)

[Structure Diagram 26](#_Toc517366952)

# Algorithms

## Rationale:

Algorithm’s are needed for planning out the actual coding in the project

## Algorithms

They are below

### ScreenMsg

ScreenMsg(screen,x,y,font,text,colour)

**BEGIN**

Text=font.render(text,colour)

Screen.blit(text,(x,y))

**END**

### Rectbutton

RectButton(screen,x,y,w,h,colour,events,text,font,fcolour)

**BEGIN**

Text=font.render(text,fcolour)

Draw.rect(screen,colour,(x,y,w,h))

rect=Rect(x,y,w,h)

screen.blit(text,(x+w/2-text.width()/2,y+h/2-text.height()/2))

**FOR** event=events[i] **TO** length(events):

**IF** event.type == Mouseclick():

**IF** rect.collision(mouse.pos())=True:

**RETURN** True

**ENDIF**

**ENDIF**

**NEXT i**

**RETURN** False

**END**

### Imgbutton

ImgButton(screen,img,x,y,events)

**BEGIN**

Screen.blit(img,(x,y))

Rect = py.rect(x,y,img.width(),img.height())

**FOR** event=events[i] **TO** length(events):

**IF** event.type == Mouseclick():

**IF** rect.collision(mouse.pos())=True:

**RETURN** True

**ENDIF**

**ENDIF**

**NEXT i**

**END**

### SelectLevel

Selectlevel()

**BEGIN**

Tick(10)  
 events=py.event.get()

**FOR** event=events[i] **TO** length(events):

**IF** event.type == Quit():

Pygame.quit()

**ENDIF**

**NEXT i**

Screen.fill(GRAY)

Counter=0

X=0

Y=0

**WHILE** X != 3:

**WHILE** Y != 5:

Counter=Counter+1

Click=rectbutton(screen,175+160\*y,300+100\*x,50,50,BLACK,events,str(cou),myfont,WHITE)

**IF** Click == True:

RETURN(Counter)

**END IF**

**END WHILE**

**END WHILE**

Pygame.display.flip()

**END**

### Gamescreen

This algorithm does the main game logic.

Gamescreen(LevelNum, LevImageRes, LevImagePlay)

**BEGIN**

Tick(100)

Global PrevClickedResImg

Global PrevClickedRes

Global CorrectSpot

Global Score

events=py.event.get()

**FOR** event=events[i] **TO** length(events):

**IF** event.type == Quit():

Pygame.quit()

**ENDIF**

**NEXT i**

Screen.fill(GRAY)

Playcount=0

Tog1 = False

J1=0

J2=0

**While** J1 != 5:

**While** J2 != 5:

IF LevImagePlay[Playcount].img\_selected == False:

Tog1=imgbutton(screen,LevImagePlay[Playcount].image[LevImagePlay[Playcount].img\_count],LevImagePlay[Playcount].x,LevImagePlay[Playcount].y,events

**END IF**

**IF** LevImagePlay[Playcount].img\_selected == True:

tog1=imgbutton(screen,GreyBox,LevImagePlay[Playcount].x,LevImagePlay[Playcount].y,events)

**END IF**

**IF**  Tog1 == True:

if LevImagePlay[Playcount].img\_count == prevclickedResImg:

LevImagePlay[Playcount].img\_selected = False

LevImageRes[prevclickedRes].correct=True

Correctspot[prevclickedRes]=True

score=score+10

**END IF**

**ELSE**:

score=score-10

**END ELSE**

J2=J2+1

**END WHILE**

J1=J1+2

**END WHILE**

Tog=False

Coun=0

I1 =0

I2=0

**WHILE** I1 != 5:

**WHILE** I2 != 5:

**IF** LevImageRes[coun].img\_selected == False:

tog=imgbutton(screen,LevImageRes[coun].image[LevImageRes[coun].img\_count],LevImageRes[coun].x,LevImageRes[coun].y,events)

**END IF**

**IF** LevImageRes[coun].img\_selected == True **OR** LevImageRes[coun].correct==True:

screen.blit(GreyBox,(LevImageRes[coun].x,LevImageRes[coun].y))

**END IF**

**IF** tog == True:

LevImageRes[prevclickedRes].img\_selected = False

prevclickedRes=coun

prevclickedResImg=LevImageRes[coun].img\_count

LevImageRes[coun].img\_selected = True

**END IF**

I2=I2+1

**END WHILE**

I1=I1+1

**END WHILE**

ScreenMsg(screen,400,100,yfont,”Logic Puzzler”,ORANGE)

ScoreMsg=”Score: “ +score

ScreenMsg(screen,100,100,myfont,ScoreMsg,BLACK)

Py.display.flip()

**IF** all(correctspot) == True:

**IF** PrevClickedRes == 26:

**RETURN**(Score)

**END IF**

PrevclickedRes = 26

**END IF**

**RETURN**(-100000000000)

**END**

Scorescreen(score,scorel)

**BEGIN**

Tick(10)

**FOR** event=events[i] **TO** length(events):

**IF** event.type == Quit():

Pygame.quit()

**ENDIF**

**NEXT i**

Screen.fill(GRAY)

I=0

**WHILE** I != 5:

screenMsg(screen,100,100\*I,yfont,scorel[i],RED)

i=i+1

**END WHILE**

BTS=Rectbutton(screen,20,10,50,15,BLACK,events,”Back to Level’s”,yfont,ORANGE)

**IF** BTS == True:

**RETURN**(“BackToLevel”)

**END IF**

**END**

### Main

This algorithm controls all the logic for switching screens and setting up the necessary data structures for the different screens.

**Main()**

**BEGIN**

**IF** event == quit:

Pygame.quit

**END IF**

Txtbx.update(event)

Txtbx.draw(screen)

Name=txtbx.value

Start=imgbutton(screen,startbutt,350,100,events)

**IF** start == True **AND** name != “”:

Select=True

**END IF**

**IF**  start == True **OR** errorcount[0] == True **AND** name == “” and errorcount[0] != 100:

Errorcount[1] = True

screenMsg(screen, screen\_width/2, screen\_height/2, yfont, “INPUT A NAME”, ORANGE)

errorcount+=1

**END IF**

**if** errorcount[0] == 1000 **AND** errorcount[1] == True:

errorcount[0]=0

errorcount[1]=False

**END IF**

**WHILE** select **AND** levnum == 16:  
 **IF** event == quit:

Pygame.quit

**END IF**

Levnum=Selectlevel()

**END WHILE**

**IF** levnum != 16:

**DEFINE** imcount as an array with 25 incremented integers in it.

Shuffle imcount

Count=0

**FOR** i **TO** 5:

**FOR** j **TO** 5:

Levimagetemp=Images()

Add image to LevImageTemp.image

Set LevImageTemp.imagecount to count

Increment LevImageTemp.x with offset of 51

Increment LevImageTemp.y with offset of 51 plus 440

LevImageTemp.img\_selected=False

LevImageTemp add to Levimageres

Count+=1

**END FOR**

**END FOR**

Count=0

**FOR** i = 0 **TO** 5:

**FOR** j = 0 **TO** 5:

LevImageTemp=images()

Add image to LevImageTemp.image

Set LevImageTemp.imagecount to count

Increment LevImageTemp.x with offset of 51 plus 400

Increment LevImageTemp.y with offset of 51 plus 440

LevImageTemp.img\_selected=False

Add Levimagetemp to Levimageply

**END FOR**

**END FOR**

**WHILE** lvnum:

Gs=Gamescreen(Levnum,Levimageres,Levimageply)

**IF** gs != -100000:

Lvum=False

Gsgo=True

Create an array of arrays from the Scores.csv file as “Data”

Check if the score is higher than atleast one of the existing scores, if so replace it and move on to next line

Save data to Scores.csv

**END IF**

**WHILE**  GSGO=true:

Scorescreen()

Update the game screen

**END**

### Mainline Algorithm

Mainline()

**BEGIN**

set framerate to 100

**IF** event = Quit:

Pygame.quit

**END IF**

Main()

# DATA DICTIONARY

## Rationale:

This is needed due to the fact that It can be easy to lose track of different variables with in the code when there’s no defined list of all the different ones.

## Data Dictionary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DATA NAME** | **DATA TYPE** | **SIZE** | **SCOPE** | **DESCRIPTION** |
| BLACK | Tuple | N/A | GLOBAL | Easy to call for black |
| BLUE | Tuple | N/A | GLOBAL | Easy to call for the colour blue |
| Butt | Rectangle object | N/A | LOCAL to imgbutton func | Allows for easier collision detection |
| Correctspot | Array | N/A | GLOBAL | Stores a list of images in the correctspot |
| Cou | Integer | N/A | LOCAL to selectlevel | Counts the amount of times the for loops have looped |
| Coun | Integer | N/A | LOCAL to Gamescreen | counts the amount of times the nested for loops have looped |
| Count | integer | N/A | LOCAL to Main | Counts how many times the nested for loops have looped |
| Data | Numpy ndimensional array | N/A | LOCAL to Main | Is a 5x1 dimensional array in order to store both names and scores next to each other |
| Errorcount | Array | N/A | GLOBAL | Used for checking if an error should be displayed or not |
| Events | Pygame event queue | N/A | LOCAL to selectlevel | Allows for getting the event queue |
| Events | Pygame event queue | N/A | LOCAL to Gamescreen | Allows for getting the event queue |
| GREEN | Tuple | N/A | GLOBAL | Easy to call for the colour green |
| GreyBox | Pygame image object | N/A | GLOBAL | Used as a placeholder for a clicked image |
| Gs | Integer | N/A | LOCAL to Main | Is the players score from the level |
| Gsgo | Boolean | N/A | LOCAL to Main | Decides if the scorescreen should be shown or not |
| Imcount | Array | N/A | LOCAL to Main | Is used for easy scrambling of the image count |
| LevFilesLoc | String | N/A | LOCAL to Main | Is the location where the file is stored |
| LevImagePlay | Array of Records | N/A | LOCAL to Gamescreen | Stores information and images required for the logic to work |
| LevImageRes | Array of Records | N/A | GLOBAL | Is the AoR where all the information about the Resource images is stored |
| LevImagesPlay | Array of Records | N/A | LOCAL to Main | Is the AoR where all the information about the play images is stored |
| LevImageTemp | Record | N/A | LOCAL to Main | Is a temp variable to allow for the creation of AoR’s |
| Levnum | Integer | N/A | LOCAL to Main | Is the level number |
| Lock | Pygame Clock object | N/A | GLOBAL | Sets the clock object in order to set framerate |
| Lvnum | Boolean | N/A | LOCAL to Main | Decides if the gamescreen should be shown or not |
| Name | String | N/A | GLOBAL | Stores the user’s name |
| Newrow | Array | N/A | LOCAL to Main | Is an array that stores the players score and name in the correct format |
| ORANGE | Tuple | N/A | GLOBAL | Easy to call for the colour orange |
| Playcount | Integer | N/A | LOCAL to Gamescreen | counts the amount of times the nested for loops have looped |
| prevclickedPlay | Integer | N/A | GLOBAL | Used for caching the latest clicked play image |
| prevclickedRes | Integer | N/A | GLOBAL | Used for caching the latest clicked resource image |
| Rect | Rectangle object | N/A | LOCAL to rect button func | Allows for easier collision detection |
| RED | Tuple | N/A | GLOBAL | Easy to call for the colour red |
| Running | Boolean | N/A | GLOBAL | Defines whether the program runs or doesn’t |
| Scomsg | Pygame text object | N/A | LOCAL to Gamescreen | Is the text object that is blitted to the screen |
| Score | Integer | N/A | GLOBAL | Stores the current user score |
| ScoreMsg | String | N/A | LOCAL to Gamescreen | Is the string that is used in making scomsg |
| Screen | Pygame display object | N/A | GLOBAL | Is the screen object |
| Screen\_height | Integer | N/A | GLOBAL | Defines screen height |
| Screen\_width | integer | N/A | GLOBAL | Defines screen width |
| Select | Boolean | N/A | LOCAL to Main | Decides if the select level should be show or not |
| Start | Imgbutton object | N/A | LOCAL to Main | Allows for easy button checking |
| Startbutt | Pygame image object | N/A | GLOBAL | Image for the start button |
| Tog | Boolean | N/A | LOCAL to Gamescreen | Is assigned to a imgbutton object to let each of the images be buttons |
| Tog1 | Boolean | N/A | LOCAL to Gamescreen | Is assigned to a imgbutton object to let each of the images be buttons |
| Txtbx | Eztext text input object | N/A | GLOBAL | Allows capture of user text input |
| WHITE | Tuple | N/A | GLOBAL | Easy to call for white |
| Xv | Text object | N/A | LOCAL to screenmsg func | Text object to be blitted |

# Source Code

Rationale:

Source code is needed due to the fact that there would be nothing with out the code

## Source Code

import pygame as py

import random as rand

import eztext

import numpy as np

py.init()

#vars below

class Images:

image=[]

x=0

y=0

img\_count=0

img\_selected=False

correct=False

class Scores:

NS=[]

screen\_width = 1000

screen\_height = 700

screen = py.display.set\_mode((screen\_width,screen\_height))

lock = py.time.Clock()

myfont = py.font.SysFont("Comic Sans MS",11)

yfont = py.font.SysFont("Comic Sans MS",20)

running=True

txtbx = eztext.Input(maxlength=10, color=(255,255,255), prompt='Name: ')

startbutt=py.image.load("startbutton.png")

SL=0

py.display.set\_caption("Logic Puzzler")

RED = (255,0,0)

BLUE = (0,0,255)

GREEN = (0,255,0)

BLACK = (0,0,0)

WHITE = (255,255,255)

ORANGE = (255,180,0)

prevclickedRes=0

prevclickedPlay=0

GreyBox=py.image.load("gray.jpg")

GreyBox=py.transform.scale(GreyBox,(50,50))

prevclickedResImg=0

Correctspot=[]

score=0

name=""

errorcount=[0,False]

for i in range(25):

Correctspot.append(False)

#vars above

def screenMsg(screen,x,y,font,text,color):

text=str(text)

xv=font.render(text,1,color)

screen.blit(xv,(x,y))

def rectbutton(screen,x,y,w,h,color,events,text,font,fcolor):

xv=font.render(text,1,fcolor)

py.draw.rect(screen,color,(x,y,w,h))

rect=py.Rect(x,y,w,h)

screen.blit(xv,(x+w/2-py.Surface.get\_width(xv)/2,y+h/2-py.Surface.get\_height(xv)/2))

for evet in events:

if evet.type == py.MOUSEBUTTONUP:

if rect.collidepoint((py.mouse.get\_pos())):

return(True)

def imgbutton(screen,img,x,y,events):

screen.blit(img,(x,y))

butt=py.Rect(x,y,img.get\_width(),img.get\_height())

for event in events:

if event.type == py.MOUSEBUTTONUP:

if butt.collidepoint((py.mouse.get\_pos())):

return(True)

return(False)

def Selectlevel():

lock.tick(10)

events=py.event.get()

for event in events:

if event.type == py.QUIT:

running=False

py.display.quit()

screen.fill((50,50,50))

cou=0

for j in range(3):

for i in range(5):

cou=cou+1

click=rectbutton(screen,175+160\*i,300+100\*j,50,50,BLACK,events,str(cou),myfont,WHITE)

if click == True:

return(cou)

py.display.flip()

return(16)

def gamescreen(levelnum,LevImageRes,LevImagePlay):

lock.tick(100)

global prevclickedResImg

global prevclickedRes

global Correctspot

global score

events = py.event.get()

for event in events:

if event.type == py.QUIT:

running=False

screen.fill((50,50,50))

Playcount=0

tog1=False

for j in range(5):

for i in range(5):

if LevImagePlay[Playcount].img\_selected == False:

tog1=imgbutton(screen,LevImagePlay[Playcount].image[LevImagePlay[Playcount].img\_count],LevImagePlay[Playcount].x,LevImagePlay[Playcount].y,events)

if LevImagePlay[Playcount].img\_selected == True:

tog1=imgbutton(screen,GreyBox,LevImagePlay[Playcount].x,LevImagePlay[Playcount].y,events)

if tog1 == True:

if LevImagePlay[Playcount].img\_count == prevclickedResImg:

LevImagePlay[Playcount].img\_selected = False

LevImageRes[prevclickedRes].correct=True

Correctspot[prevclickedRes]=True

score=score+10

else:

score=score-10

Playcount+=1

coun=0

tog=False

for j in range(5):

for i in range(5):

if LevImageRes[coun].img\_selected == False:

tog=imgbutton(screen,LevImageRes[coun].image[LevImageRes[coun].img\_count],LevImageRes[coun].x,LevImageRes[coun].y,events)

if LevImageRes[coun].img\_selected == True or LevImageRes[coun].correct==True:

screen.blit(GreyBox,(LevImageRes[coun].x,LevImageRes[coun].y))

if tog == True:

LevImageRes[prevclickedRes].img\_selected = False

prevclickedRes=coun

prevclickedResImg=LevImageRes[coun].img\_count

LevImageRes[coun].img\_selected = True

coun+=1

screenMsg(screen,400,100,yfont,"LOGIC PUZZLER",ORANGE)

Scoremsg="Score: "+str(score)

Scoremsg=str(Scoremsg)

scomsg=myfont.render(Scoremsg,1,RED)

screen.blit(scomsg,(100,100))

py.display.flip()

score-=0.01

if all(Correctspot):

if prevclickedRes == 26:

return(score)

prevclickedRes=26

return(-100000000000)

def scorescreen(score,data):

lock.tick(10)

events=py.event.get()

for event in events:

if event.type == py.QUIT:

running=False

py.display.quit()

Scorerank=['5th: ','4th: ','3rd: ','2nd: ','1st: ']

screen.fill((50,50,50))

i=5

while i != 0:

temp=data[i-1]

screenMsg(screen,100,350-50\*i,yfont,Scorerank[i-1]+str(temp[0]),RED)

screenMsg(screen,200,350-50\*i,yfont,temp[1],RED)

i=i-1

py.display.flip()

def main():

global name

global errorcount

events=py.event.get()

for event in events:

if event.type == py.QUIT:

running=False

screen.fill((50,50,50))

select=False

lvnum=False

levnum=16

gsgo=False

txtbx.update(events)

#blit txtbx on the screen

txtbx.set\_pos(100,10)

txtbx.draw(screen)

name=txtbx.value

start=imgbutton(screen,startbutt,350,100,events)

if start == True and name !='':

select=True

if start == True or errorcount[1]==True and name == '' and errorcount[0] !=100:

errorcount[1]=True

screenMsg(screen,screen\_width/2,screen\_height/2,yfont,"INPUT A NAME",ORANGE)

errorcount[0]+=1

if errorcount[0] == 1000 and errorcount[1] == True:

errorcount[0]=0

errorcount[1]=False

while select and levnum == 16:

for event in py.event.get():

if event.type == py.QUIT:

running=False

levnum=Selectlevel()

if levnum != 16:

imcount=[]

for i in range(25):

imcount.append(i)

rand.shuffle(imcount)

count=0

LevFilesLoc="Levels/"+str(levnum)+"/"

LevImageRes=[]

for j in range(5):

for i in range(5):

LevImagesTemp=Images()

img=py.image.load(LevFilesLoc +str(count+1) +".jpg").convert\_alpha()

img=py.transform.scale(img,(50,50))

LevImagesTemp.img\_count=imcount[count]

LevImagesTemp.image.append(img)

LevImagesTemp.x=i\*51

LevImagesTemp.y=j\*51+440

LevImagesTemp.img\_selected=False

LevImageRes.append(LevImagesTemp)

count+=1

count=0

LevImagesPlay=[]

for j in range(5):

for i in range(5):

LevImagesTemp=Images()

img=py.image.load(LevFilesLoc +str(count+1) +".jpg").convert\_alpha()

img=py.transform.scale(img,(50,50))

LevImagesTemp.img\_count=count

LevImagesTemp.image.append(img)

LevImagesTemp.x=i\*51+400

LevImagesTemp.y=j\*51+440

LevImagesTemp.img\_selected=True

LevImagesTemp.correct=False

LevImagesPlay.append(LevImagesTemp)

count+=1

lvnum = True

while lvnum:

gs=gamescreen(levnum,LevImageRes,LevImagesPlay)

if gs != -100000000000:

lvnum=False

gsgo=True

data = np.genfromtxt('Scores.csv', delimiter=',', dtype=None, names=('Scores','Name'),encoding=None)

newrow=(gs,name)

checker=0

for i in range(5):

temp=data[i]

if newrow[0] > temp[0] and checker == 0:

data[i]=newrow

checker+=1

data.sort(order='Scores')

np.savetxt("Scores.csv",data,fmt=('%i, %s'),delimiter=",")

while gsgo:

scorescreen(gs,data)

py.display.flip()

while running:

lock.tick(100)

events=py.event.get(py.QUIT)

for event in events:

if event.type == py.QUIT:

running=False

main()

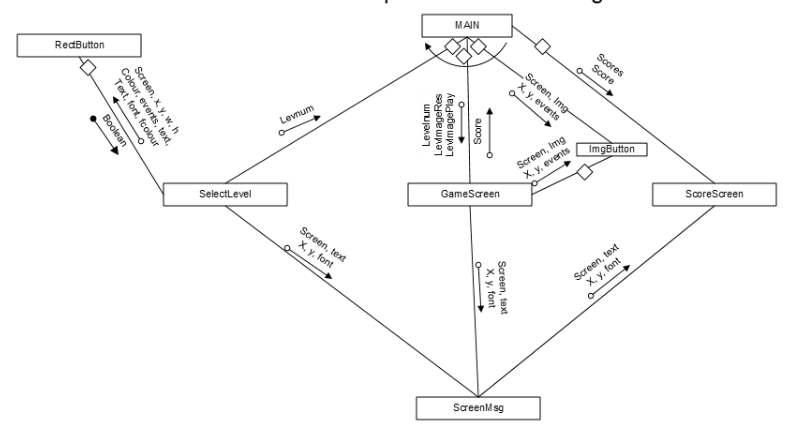
#xkcc

# Structure Diagram

## Rationale:

This is needed to easily see how data flows through and around the program through between the different functions

## Structure Diagram



# Logbooks

## Rationale:

This is needed due to the fact that keeping a track of how the developer has used their time and if they’ve actually been working on it

## Logbook

### 26/03/18

Today I’ve started working on the major work, I have decided to create a logic puzzle game with dynamically updating gates and with easy to read code with comments in it. I’ve also made the Wing IDE commit work from my laptop. This took a surprising amount of time due to me forgetting to add a commit message to the commit

### 27/03/18

I’ve made Wing IDE commit work on my desktop, this was much easier than on my laptop seeming as I have realized how to put a commit message in the commit box. I also made my image button function tonight which currently works fine. I also added in the eztext file which I use for text input for most of my pygame projects. I also started work on functionalizing the code in order to make it be easier to read. I also added in the start button image and added more screens to the program

### 30/03/18

Today I have made progress on the Level selection screen. Currently it just changes to it and then nothing happens.

### 31/03/18

Today I have added in buttons, however currently only one button works. Of course it is the first one to be blitted.

### 05/04/18

Today I made ALL OF THE BUTTONS WORK. The solution to this was parsing the pygame events queue into the button function rather than just having it be a global variable. This made me be very happy

### 07/04/18

Today I’ve added in the rest of the buttons and also made it so that they also have the text that I want centered in them. The centering of the text actually took longer than I expected. This is due to me using trial and error rather than actually working it out before.

### 09/04/18

Today I’ve added in the screenMsg and togglerectbutton functions. The screenMsg function is so I can easily blit text onto the screen. The togglerectbutton function is used to have a button that can be toggled easily.

### 10/04/18

Today I added in the images needed for the first level and also moved away from using a modular logic gate system and moved to just a normal puzzle game functionality now seeming as it was going to be too complex to do the dynamic logic gate puzzle.

### 12/04/18

Today I axed the togglerectbutton function seeming as it was giving me too much grief over actually getting it working in time with the proper functionality. I also started work on the data structures that I’d need for the final logic.

### 13/04/18

Today I added the images class and also the text input. I also started using the class to make an array of records to more easier make the data manipulation more reasonable and logical

### 02/05/18

Today I worked on adding the other other array of records for the other section of the gamescreen. I also started setting up the logic for the actual game.

### 07/05/18

Today I worked on adding the actual puzzle functionality into the game. This is one of the largest additions since the beginning seeming as there is a large amount of logic to get working. I actually managed to have this be done within the lesson which I’m very happy about.

### 08/05/18

Today I added in more of the logic needed for the puzzle game getting the logic to this point has actually been quite challenging even using my previous solitare game to help me with designing the logic.

### 11/05/18

This time I just worked on cleaning up my code and the actual amount of files that were being used in the file structure. I also added in an extra 25 files for the second level and I still need to add the other 13 levels. With 25 files per level I have 325 files left to add just for the levels at this point.

### 13/05/18

Today I decided to add PyIgnition to make the scorescreen be more cool with some dank particle effects

### 21/05/18

Today I actually added in the scoring functionality how ever it is sadly not completely non-functional right now. Today I also decided to remove PyIgnition due to it not being in keeping with the general design aesthetic.

### 24/05/18

Today I started working on the algorithms and also on making the ability to make images go transparent.

### 27/05/18

Today I FINISHED THE ALGORITHMS!

### 04/06/18

Changed the scores file to a csv to to try and make score input work better and be easier to manipulate.

### 05/06/18

Today I worked on using the CSV library some more, still couldn’t get it to work somehow still need to do more work on this part of the code seeming as it isn’t working.

### 07/06/18

Today I decided to instead of using the csv module to instead use NUMPY which allows me to easily take a file and generate an array of arrays which I find to be able to manipulate easier. I also made it so that it can load in the file however I currently cannot save it to a file.

### 08/06/18

Today I made it so that I can now save the scores to the file and also save it to the same file. This took a significant amount of work seeming as NUMPY’s documentation for the formatting being severely subpar. It took me eventually guessing the correct formatting option to make it work.

### 19/06/18

Today I updated my algorithms a bit more

### 20/06/18

Today I just updated the scorescreen and updated it.

### 21/06/18

Today I FINISHED my data dictionary and added my structure diagram with help from Mr Thill.