





***ACKNOWLEDGEMENT:***

“Alone we can do so little, together we can do so much.” – Helen Keller

Acknowledgement is the most beautiful page in any project’s final pages. More than a formality, this appears to me the best opportunity to express our gratitude.

We have indeed put our share of efforts in this project. But it would never have been as fruitful without the support of all those who made this project a success.

We would like to say a big thank you from the bottom of our heart to the following people. Firstly, I would like to thank the school management and our principal – Mrs. Shantha Chandran for providing us an apt environment to accomplish the task. Next, I would like to express my gratitude to my teachers – Mrs. Manjula and Mrs. Suguna for their patient expert guidance and willingness to help whenever required. Last but not the least, We also appreciate the help provided by our parents in completing this project.

***INTRODUCTION :***

Our project is a game based on an existing game called ‘Cut the Rope’. The graphics of the game have been incorporated with the help of an external module, Pygame. It involves real life simulations like gravity and oscillations. The objective of this game is to get the candy to the bowl at the bottom of the screen, by cutting a rope, as the name suggest, and getting past some obstacles. On the way, there are stars for bonus points. All you have to do is to cut the rope ,collect all the stars, burst the bubble and escape traps to reach the bowl in the shortest time possible. We have incorporated the concepts of classes and objects, file handling, exception handling.



**SYSTEM REQUIREMENTS :**

*Hardware Requirements*:

* Pentium processor/higher with 2 GB RAM
* Mouse
* Keyboard
* Colour monitor

*Software Requirements****:***

* Operating system – Windows XP or higher
* Python version 2.7.4 or higher
* 3D OpenGL compatible graphics accelerator card.
* 256 MB RAM.
* 500 Mhz Pentium 3 processor.

*Recommendations:*

* A late-model 3D OpenGL compatible graphics accelerator card from nVidia, ATI, 3Dlabs or similar.
* 512 MB RAM (768 MB or 1 GB preferred).
* 3 Ghz Pentium 4 processor or similar.

**MODULES AND FUNCTIONS USED :**

*Modules:*

* pygame module
* math module
* Time module

*Library functions:*

* pygame init() – to initialize the pygame module.
* mixer.init() – to initialize the mixer sound options.
* mixer.music.load(file) – to load the playback music file.
* mixer.music.play(file) – to play the music loaded.
* mixer.music.get\_busy() – to check whether the mixer is in use.
* mixer.music.stop() – to stop playing the playback music.
* surface.blit() – to render the background object on the screen.
* display.set\_mode((x,y)) –to create a window of required size.
* display.set\_caption(caption) –to set the title for the window created.
* display.update() – to update the screen from time to time.
* time.Clock() – to have an access to time.
* image.load(image) – to load the image required as an object.
* pygame.transform.scale(object,(x,y)) – to transform the image to a required size.
* event.get() – to get the state of various input devices.
* mouse.get\_pos() – to get the position of mouse at any instant of time.
* draw.circle(screen,color,(x,y)<center>,radius,thickness) – to draw a circle of required dimension on the screen.
* draw.lines(screen,color,closed<True/False>,pointlist,thickness) – to draw lines on the screen.
* quit() – to exit from the window created.
* math.sin(angle) – to obtain sine of an angle.
* math.cos(angle) – to obtain cosine of an angle.
* time. sleep() – to make the delay.

**GAME FLOWCHART :**





***USER DEFINED FUNCTIONS AND DATA TYPES USED IN THE CODE :***

1. *Sound():* It takes no parameters. It is evoked to control the play back music options such as on and off.
2. *Menu():* It takes no parameters. It is evoked to control and to facilitate the main menu options such as play and quit.
3. *LevelSelect():* It takes no parameters. It is evoked to control and facilitate the level selection options.
4. *swing():* It takes 6 parameters. It is responsible for the swinging action of the candy. It returns the angle with vertical, tangential velocities of the candy and the new position of candy each time this function is invoked.
5. *Result():* It takes 1 parameter. This function calculates the score and displays the number of stars collected if won. If lost , allows you to try again .
6. *Fall():* It takes 13 parameters in which 3 are default parameters. This function takes the responsibility of motion of candy after being cut. It takes care of the normal freefall after being cut ,the bubble floating up, checking whether the bubble gets popped, checking for stars and check whether the bowl has been reached or whether the target has been missed. It returns a Boolean value True or False whether the target has been reached or not .
7. *check\_star()*: It takes 2 parameters. This is a function that checks whether the star has been caught by the candy or not. It does not return any value.
8. blit():
9. *cut():* It takes 4 parameters. This function checks whether the rope of the candy has been cut or not. It returns a Boolean value True if cut, otherwise False.
10. *Game\_Loop()*: It takes 9 parameters. This function is the main game loop that handles the levels of the game. It does not return any value.
11. *render\_textrect():* Returns a surface containing the passed text string, reformatted to fit within the given rectangle , word-wrapping as necessary. The text will be anti-aliased.

**VARIABLES USED AND THEIR DATATYPES:**

|  |  |
| --- | --- |
| VARIABLE | DATATYPE |
| file | string |
| white | tuple |
| black | tuple |
| blue | tuple |
| red | tuple |
| display\_h | int |
| candy\_width | int |
| display\_w | int |
| mx | float |
| my | float |
| x | float |
| y | float |
| cx | float |
| cy | float |
| r | float |
| ay | float |
| t | float |
| angle | float |
| angle\_max | float |
| yn | float |
| xn | float |
| vx | float |
| vy | float |
| win | boolean |
| time\_taken | float |
| num\_stars | int |
| scor | int |
| font\_size | int |
| a | float |
| x\_bowl | float |
| y\_bowl | float |
| cont | boolean |
| h | float |
| occupied | boolean |
| exist | boolean |
| x\_centre | float |
| y\_centre | float |
| r\_candy | float |
| x\_bowl | float |
| y\_bowl | float |
| acc | int |
| dt | float |
| final\_lines | list |
| words | list |
| c | int |
| n | string |
| accumulated\_line | string |
| l1 | Object of class levels |
| l2 | Object of class levels |
| l3 | Object of class levels |
| check | int |
| S1 | Object of class stars |
| S2 | Object of class stars |
| S3 | Object of class stars |

**SOURCE CODE :**

# importing all required modules

import pygame

import math

import time

import pickle

#initialising pygame

pygame.init()

file = '123.mp3'

pygame.mixer.init()

pygame.mixer.music.load(file)

pygame.mixer.music.play()

#colours

white = (255,255,255)

black = (0,0,0)

red = (255,0,0)

blue = (0,0,255)

#etc

# max height is, say, 2000 (for window)

display\_h = 800

display\_w = int(display\_h\*1.2)

# creating the screen window

surface = pygame.display.set\_mode((display\_w,display\_h))

pygame.display.set\_caption("Cut The Rope")

#creating clock to track time

clock=pygame.time.Clock()

# note : max img size=2000/20

candy\_width = int(display\_h/20)

#creating objects for the images required

candy = pygame.image.load('CANDY.png')

bowl=pygame.image.load('bowl1.png')

##bwc=pygame.image.load('bowl and candy.png')

stari=pygame.image.load('star2.png')

menu = pygame.image.load('menu.png')

back = pygame.image.load('back6.png')

back1 = pygame.image.load('back1.png')

sound\_on = pygame.image.load('sound\_on.png')

sound\_off = pygame.image.load('sound\_off.png')

levelselect = pygame.image.load('level select.png')

score1 = pygame.image.load('s1.png')

score2 = pygame.image.load('s2.png')

score3 = pygame.image.load('s3.png')

score0 = pygame.image.load('s0.png')

go2menu = pygame.image.load('go2menu.png')

playagain = pygame.image.load('playagain.png')

nextlevel = pygame.image.load('nextlevel.png')

hat1 = pygame.image.load('redhat.png')

hat2 = pygame.image.load('bluehat.png')

#transforming images to the sizes required

candy = pygame.transform.scale(candy,(candy\_width,candy\_width))

bowl = pygame.transform.scale(bowl,((candy\_width\*4,candy\_width\*4)))

stari= pygame.transform.scale(stari,(candy\_width,candy\_width))

menu = pygame.transform.scale(menu,(display\_w,display\_h))

back = pygame.transform.scale(back,(display\_w,display\_h))

back1 = pygame.transform.scale(back1,(display\_w,display\_h))

sound\_on = pygame.transform.scale(sound\_on,(int(display\_w\*0.30576923),int(display\_h\*0.1358975)))

sound\_off = pygame.transform.scale(sound\_off,(int(display\_w\*0.30576923),int(display\_h\*0.1358975)))

levelselect = pygame.transform.scale(levelselect,(int(display\_h\*0.7),int(display\_h\*0.8)))

score1 = pygame.transform.scale(score1,(int(candy\_width\*6),int(candy\_width\*1.6236)))

score2 = pygame.transform.scale(score2,(int(candy\_width\*6),int(candy\_width\*1.6236)))

score3 = pygame.transform.scale(score3,(int(candy\_width\*6),int(candy\_width\*1.6236)))

score0 = pygame.transform.scale(score0,(int(candy\_width\*6),int(candy\_width\*1.6236)))

go2menu = pygame.transform.scale(go2menu,(int(candy\_width\*4),int(candy\_width\*2)))

nextlevel = pygame.transform.scale(nextlevel,(int(candy\_width\*4),int(candy\_width\*2)))

playagain = pygame.transform.scale(playagain,(int(candy\_width\*4),int(candy\_width\*2)))

hat1 = pygame.transform.scale(hat1,(int(candy\_width\*2),int(candy\_width\*2)))

hat2 = pygame.transform.scale(hat2,(int(candy\_width\*2),int(candy\_width\*2)))

hat1 = pygame.transform.rotate(hat1,170)

#001

#For background sound options ON or OFF

def Sound():

if pygame.mixer.music.get\_busy():

surface.blit(menu, (0, 0))

surface.blit(sound\_off,(display\_h/2.425373,display\_h/1.340206))

pygame.mixer.music.stop()

pygame.display.update()

else:

surface.blit(menu, (0, 0))

surface.blit(sound\_on,(display\_h/2.425373,display\_h/1.340206))

pygame.mixer.music.play()

pygame.display.update()

#002

#For managing the menu window

def Menu():

surface.blit(menu, (0, 0))

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

surface.blit(sound\_on,(int(display\_h/2.425373),int(display\_h/1.340206)))

pygame.display.update()

global sound\_state

c = 0

while 1:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

if event.type == pygame.MOUSEBUTTONUP:

(mx,my) = pygame.mouse.get\_pos()

if int(mx) in range(int(display\_h/2.32142857),int(display\_h/1.3)) and int(my) in range(int(display\_h/1.3131313),int(display\_h/1.17117117)):

Sound() # calling 001

if int(mx) in range(int(display\_h/2.32142857),int(display\_h/1.3)) and int(my) in range(int(display\_h/1.547619),int(display\_h/1.35416667)):

c = 1

break

if int(mx) in range(int(display\_h/1.2),int(display\_h/1.05)) and int(my) in range(int(display\_h/1.547619),int(display\_h/1.35416667)):

print 'here'

fo=open('Highscores.DAT','rb')

L=[]

try:

L+=[pickle.load(fo)]

except:

fo.close()

for i in L:

print i.name,'\t',i.score

if c:

break

LevelSelect()

# to handle level selection

def LevelSelect():

global level

surface.blit(levelselect,(int(display\_h\*0.6-display\_h\*0.35),display\_h\*0.1))

pygame.display.update()

while 1:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

if event.type == pygame.MOUSEBUTTONUP:

(mx,my) = pygame.mouse.get\_pos()

if int(mx) in range(int(display\_h/2.15),int(display\_h/1.36)) and int(my) in range(int(display\_h/5),int(display\_h/3.6)):

l1.time\_start = time.time()

level = l1

l1.Play\_Level()

if int(mx) in range(int(display\_h/2.15),int(display\_h/1.36)) and int(my) in range(int(display\_h/2.5),int(display\_h/2.07)):

l2.time\_start = time.time()

level = l2

l2.Play\_Level()

if int(mx) in range(int(display\_h/2.15),int(display\_h/1.36)) and int(my) in range(int(display\_h/1.67),int(display\_h/1.46)):

l3.time\_start = time.time()

level = l3

l3.Play\_Level()

#003

#For swinging movement of the candy

def swing((x,y), (cx,cy), r, ay, dt, t):

global check

global angle\_max

angle\_max \*= 0.9995

angle = angle\_max \* math.sin((ay/r)\*\*1/2 \* (t+2.5))

yn = r \* math.cos(angle) + cy

xn = r \* math.sin(angle) + cx

vx =(xn - x)/dt

vy = (yn - y)/dt

return (xn,yn),angle,[vx,vy]

#004

# to calculate scorecard if won

def Result(win):

time.sleep(1.5)

global level

if win:

time\_taken = time.time()-level.time\_start

num\_stars = 0

for i in [s1,s2,s3]:

if not i.exist:

num\_stars += 1

scor = int(10\*(200+200 \* num\_stars )/ time\_taken)

scor = score(scor)

scor.check\_high()

font\_size = pygame.font.Font('freesansbold.ttf',int(candy\_width/2))

WIN = font\_size.render('YOU WIN! CONGRATS =)', True, black)

scortxt = font\_size.render('Score: '+str(scor.score),True,black)

WIN\_Rect = WIN.get\_rect()

WIN\_Rect.center = (int(display\_w/2), int(display\_h/2-candy\_width\*3))

score\_Rect = scortxt.get\_rect()

score\_Rect.center = ((display\_w/2), (display\_h/2-candy\_width\*2))

surface.blit(back,(0,0))

surface.blit(WIN, WIN\_Rect)

surface.blit(scortxt,score\_Rect)

if num\_stars == 0:

surface.blit(score0,(int(display\_w/2-candy\_width\*3), int(display\_h/2-candy\_width)))

elif num\_stars == 1:

surface.blit(score1,(int(display\_w/2-candy\_width\*3), int(display\_h/2-candy\_width)))

elif num\_stars == 2:

surface.blit(score2,(int(display\_w/2-candy\_width\*3), int(display\_h/2-candy\_width)))

elif num\_stars == 3:

surface.blit(score3,(int(display\_w/2-candy\_width\*3), int(display\_h/2-candy\_width)))

if level.level == 3:

surface.blit(playagain,(int(display\_w/2-candy\_width\*2), int(display\_h/2+candy\_width\*3)))

surface.blit(go2menu,(int(display\_w/2-candy\_width\*2), int(display\_h/2+candy\_width\*5)))

pygame.display.update()

while 1:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

if event.type == pygame.MOUSEBUTTONUP:

(mx,my) = pygame.mouse.get\_pos()

if int(mx) in range(int(display\_w/2-candy\_width\*2),int(display\_w/2+candy\_width\*2)):

if int(my) in range(int(display\_h/2+candy\_width\*3),int(display\_h/2+candy\_width\*5)):

level.time\_start = time.time()

level.Play\_Level()

if int(mx) in range(int(display\_w/2-candy\_width\*2),int(display\_w/2+candy\_width\*2)):

if int(my) in range(int(display\_h/2+candy\_width\*5),int(display\_h/2+candy\_width\*7)):

Menu()

else:

surface.blit(playagain,(int(display\_w/2-candy\_width\*4.5), int(display\_h/2+candy\_width\*3)))

surface.blit(nextlevel,(int(display\_w/2+candy\_width\*0.5), int(display\_h/2+candy\_width\*3)))

surface.blit(go2menu,(int(display\_w/2-candy\_width\*2), int(display\_h/2+candy\_width\*5)))

pygame.display.update()

while 1:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

if event.type == pygame.MOUSEBUTTONUP:

(mx,my) = pygame.mouse.get\_pos()

if int(mx) in range(int(display\_w/2-candy\_width\*4.5),int(display\_w/2-candy\_width\*0.5)):

if int(my) in range(int(display\_h/2+candy\_width\*3),int(display\_h/2+candy\_width\*5)):

level.time\_start = time.time()

level.Play\_Level()

if int(mx) in range(int(display\_w/2+candy\_width\*0.5),int(display\_w/2+candy\_width\*4.5)):

if int(my) in range(int(display\_h/2+candy\_width\*3),int(display\_h/2+candy\_width\*5)):

if level.level == 1:

level = l2

elif level.level == 2:

level = l3

level.time\_start = time.time()

level.Play\_Level()

if int(mx) in range(int(display\_w/2-candy\_width\*2),int(display\_w/2+candy\_width\*2)):

if int(my) in range(int(display\_h/2+candy\_width\*5),int(display\_h/2+candy\_width\*7)):

Menu()

else:

surface.blit(back,(0,0))

font\_size = pygame.font.Font('freesansbold.ttf',candy\_width/2)

LOSE = font\_size.render('Oops. Try again!', True, black)

LOSE\_Rect = LOSE.get\_rect()

LOSE\_Rect.center = ((display\_w/2), (display\_h/2 - candy\_width\*2))

surface.blit(LOSE, LOSE\_Rect)

surface.blit(playagain,(int(display\_w/2-candy\_width\*2), int(display\_h/2)))

surface.blit(go2menu,(int(display\_w/2-candy\_width\*2), int(display\_h/2+candy\_width\*2)))

pygame.display.update()

while 1:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

if event.type == pygame.MOUSEBUTTONUP:

(mx,my) = pygame.mouse.get\_pos()

if int(mx) in range(int(display\_w/2-candy\_width\*2),int(display\_w/2+candy\_width\*2)):

if int(my) in range(int(display\_h/2),int(display\_h/2+candy\_width\*2)):

level.time\_start = time.time()

level.Play\_Level()

if int(mx) in range(int(display\_w/2-candy\_width\*2),int(display\_w/2+candy\_width\*2)):

if int(my) in range(int(display\_h/2+candy\_width\*2),int(display\_h/2+candy\_width\*4)):

Menu()

#005

# to handle the motion of candy after being cut

def fall((x,y),angle, v, a, dt, t,x\_bowl,y\_bowl, xc, yc, xc2=0, yc2=0, r2=0):

global s1,s2,s3,r,h1,h2,hi1,hi2

cont = True

t\_h = 0

while cont:

c=0

#For normal freefall, not within bubble - r (r for ring)

if r.occupied==False:

x += v[0]\*dt

y += v[1]\*dt

v[1] += a\*dt

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

#For bubble floating up

else:

v0=200 #some const speed in bubble

y -= 200\*dt

r.y=y+5

v[0] = 0

v[1] = 0

#checking if bubble is getting popped

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

elif event.type == pygame.MOUSEBUTTONUP:

pos = pygame.mouse.get\_pos()

if int(pos[0]) in range(int(r.x-candy\_width/2-5),int(r.x+candy\_width/2+5)):

if int(pos[1]) in range(int(r.y-candy\_width/2-5),int(r.y+candy\_width/2+5)):

r.occupied = False

r.exist = False

blit(back,(x,y),(x\_bowl,y\_bowl),(xc,yc),False,xc2,yc2,v)

pygame.display.update()

clock.tick(100)

#falling out of screen

if y > display\_h+candy\_width or x > display\_w+candy\_width or x < -candy\_width or y < -candy\_width:

cont = False

Result(False)

t+=dt

#checking for stars

check\_star(x,y)

#check for hat

for h in (h1,h2,hi1,hi2):

if h.exist:

if int(x) in range(h.pos1[0]-candy\_width,h.pos1[0]+candy\_width\*2):

if int(y) in range(h.pos1[1]-candy\_width,h.pos1[1]+candy\_width\*2):

if h.d1 == 'u' and v[1]>0:

surface.blit(h.img1,h.pos1)

pygame.display.update()

if t>t\_h+0.5 and int(x) in range(int(h.pos1[0]-candy\_width\*0.2),int(h.pos1[0]+candy\_width\*1.2)):

if int(y) in range(h.pos1[1],h.pos1[1]+candy\_width):

t\_h = t

(x,y) = h.pos2

if h.d2 == 'u':

v[0],v[1] = 0,-v[1]

if h.d2 == 'l':

v[0],v[1] = -v[1],0

if h.d2 == 'r':

v[0],v[1] = v[1],0

else:

v[0],v[1] = 0,v[1]

if h.d1 == 'l' and v[0]>0:

surface.blit(h.img1,h.pos1)

pygame.display.update()

if t>t\_h+0.5 and int(x) in range(h.pos1[0],h.pos1[0]+candy\_width):

if int(y) in range(int(h.pos1[1]-candy\_width\*0.2),int(h.pos1[1]+candy\_width\*1.2)):

t\_h = t

(x,y) = h.pos2

if h.d2 == 'u':

v[0],v[1] = 0,-v[0]

if h.d2 == 'l':

v[0],v[1] = -v[0],0

if h.d2 == 'd':

v[0],v[1] = 0,v[0]

else:

v[0],v[1] = v[0],0

if h.d1 == 'r' and v[0]<0:

surface.blit(h.img1,h.pos1)

pygame.display.update()

if t>t\_h+0.5 and int(x) in range(h.pos1[0],h.pos1[0]+candy\_width):

if int(y) in range(int(h.pos1[1]-candy\_width\*0.2),int(h.pos1[1]+candy\_width\*1.2)):

t\_h = t

(x,y) = h.pos2

if h.d2 == 'u':

v[0],v[1] = 0,v[0]

if h.d2 == 'd':

v[0],v[1] = 0,-v[0]

if h.d2 == 'r':

v[0],v[1] = -v[0],0

else:

v[0],v[1] = v[0],0

if h.d1 == 'd' and v[1]<0:

surface.blit(h.img1,h.pos1)

pygame.display.update()

if t>t\_h+0.5 and int(x) in range(int(h.pos1[0]-candy\_width\*0.2),int(h.pos1[0]+candy\_width\*1.2)):

if int(y) in range(h.pos1[1],h.pos1[1]+candy\_width):

t\_h = t

(x,y) = h.pos2

if h.d2 == 'd':

v[0],v[1] = 0,-v[1]

if h.d2 == 'l':

v[0],v[1] = v[1],0

if h.d2 == 'r':

v[0],v[1] = -v[1],0

else:

v[0],v[1] = 0,v[1]

#check for bubble if not already inside

if r.occupied == False and r.exist == True:

for i in range(int(r.x-candy\_width\*1.8), int(r.x+candy\_width\*0.8)):

if i==int(x):

for j in range(int(r.y-candy\_width\*0.8), int(r.y+candy\_width\*0.8)):

if j==int(y):

r.occupied = True

x,y=r.x-candy\_width/2,r.y-candy\_width/2

#check for bowl

for i in range(int(x\_bowl-candy\_width/2),int(x\_bowl+candy\_width\*2.8)):

if i==int(x):

for i in range(int(y\_bowl+candy\_width\*0.2),y\_bowl+candy\_width):

if i==int(y):

c=1

if c==1:

cont=False

Result(True)

#function to check if candy hits star

def check\_star(x,y):

global s1,s2,s3

for s in [s1,s2,s3]:

for i in range(int(s.x-candy\_width\*0.8), int(s.x+candy\_width\*0.8)):

if i==int(x):

for j in range(int(s.y-candy\_width\*0.8), int(s.y+candy\_width\*0.8)):

if j==int(y):

s.exist = False

def blit(back,(x,y),(xb,yb),(xc,yc),sw=False,xc2=0,yc2=0,v=[0,0]):

global s1,s2,s3,r,h1,h2

surface.blit(back,(0,0))

surface.blit(bowl,(xb,yb))

if s1.exist==True:

surface.blit(s1.img,(s1.x,s1.y))

if s2.exist==True:

surface.blit(s2.img,(s2.x,s2.y))

if s3.exist==True:

surface.blit(s3.img,(s3.x,s3.y))

if r.exist==True:

pygame.draw.circle(surface,blue,(int(r.x),int(r.y)),int(candy\_width\*0.5+30),10)

if sw==True:

pygame.draw.lines(surface, black, False,[(x+candy\_width/2, y+candy\_width/2),(xc+candy\_width/2, yc+candy\_width/2)], 2)

pygame.draw.circle(surface,black,[xc+20,yc+20],5,5)

if xc2 and yc2:

pygame.draw.circle(surface,black,[xc2+20,yc2+20],5,5)

if h1.exist:

surface.blit(h1.img1,h1.pos1)

surface.blit(h1.img2,h1.pos2)

if h2.exist:

surface.blit(h2.img1,h2.pos1)

surface.blit(h2.img2,h2.pos2)

surface.blit(candy,(x,y))

#006

# to check whether the rope is cut or not

def cut(x\_candy,y\_candy,xc,yc):

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

elif event.type == pygame.MOUSEMOTION:

pygame.mouse.set\_visible(True)

ctr=0

(x,y)=event.pos

if x\_candy>xc:

for i in range(xc,int(x\_candy)):

if i==x:

ctr=1

elif x\_candy<xc:

for i in range(int(x\_candy),xc):

if i==x:

ctr=1

if ctr==1:

for i in range(yc,int(y\_candy)):

if i==y:

ctr=2

if ctr==2:

return True

else:

return False

else:

return False

#007

# main game loop

def Game\_Loop(l, x\_center, y\_center, r\_candy, x\_bowl, y\_bowl, x\_center2, y\_center2, r\_candy2):

global angle\_max

global s1,s2,s3

angle\_max = (math.pi/2)

x\_candy = x\_center + r\_candy

y\_candy = y\_center

dt = 0.01

t = 0

angle = 0

vel=[0,0]

if l == 1:

acc = 800

elif l == 2:

acc = 1000

elif l == 3:

acc = 1100

else:

print "ERROR: unidentifyable level"

time.sleep(5)

while 1:

check\_star(x\_candy,y\_candy)

if cut(x\_candy,y\_candy,x\_center,y\_center):

break

else:

((x\_candy, y\_candy), angle, vel) = swing((x\_candy,y\_candy),(x\_center, y\_center), r\_candy, acc, dt, t)

blit(back,(x\_candy,y\_candy),(x\_bowl,y\_bowl),(x\_center,y\_center),True,x\_center2,y\_center2)

pygame.display.update()

clock.tick(100)

t += dt

fall((x\_candy, y\_candy),angle, vel, acc, dt, t, x\_bowl,y\_bowl,x\_center,y\_center, x\_center2, y\_center2, r\_candy2)

class TextRectException:

def \_\_init\_\_(self, message = None):

self.message = message

def \_\_str\_\_(self):

return self.message

def render\_textrect(string, font, rect, text\_color, background\_color, justification=0):

"""Returns a surface containing the passed text string, reformatted

to fit within the given rect, word-wrapping as necessary. The text

will be anti-aliased.”””

“””Takes the following arguments:

string - the text you wish to render. \n begins a new line.

font - a Font object

rect - a rectstyle giving the size of the surface requested.

text\_color - a three-byte tuple of the rgb value of the

text color. ex (0, 0, 0) = BLACK

background\_color - a three-byte tuple of the rgb value of the surface.

justification - 0 (default) left-justified

1 horizontally centered

2 right-justified

Returns the following values:

Success - a surface object with the text rendered onto it.

Failure - raises a TextRectException if the text won't fit onto the surface.

"""

import pygame

final\_lines = []

requested\_lines = string.splitlines()

# Create a series of lines that will fit on the provided

# rectangle.

for requested\_line in requested\_lines:

if font.size(requested\_line)[0] > rect.width:

words = requested\_line.split(' ')

# if any of our words are too long to fit, return.

for word in words:

if font.size(word)[0] >= rect.width:

raise TextRectException, "The word " + word + " is too long to fit in the rect passed."

# Start a new line

accumulated\_line = ""

for word in words:

test\_line = accumulated\_line + word + " "

# Build the line while the words fit.

if font.size(test\_line)[0] < rect.width:

accumulated\_line = test\_line

else:

final\_lines.append(accumulated\_line)

accumulated\_line = word + " "

final\_lines.append(accumulated\_line)

else:

final\_lines.append(requested\_line)

# Let's try to write the text out on the surface.

surface = pygame.Surface(rect.size)

surface.fill(background\_color)

accumulated\_height = 0

for line in final\_lines:

if accumulated\_height + font.size(line)[1] >= rect.height:

raise TextRectException, "Once word-wrapped, the text string was too tall to fit in the rect."

if line != "":

tempsurface = font.render(line, 1, text\_color)

if justification == 0:

surface.blit(tempsurface, (0, accumulated\_height))

elif justification == 1:

surface.blit(tempsurface, ((rect.width - tempsurface.get\_width()) / 2, accumulated\_height))

elif justification == 2:

surface.blit(tempsurface, (rect.width - tempsurface.get\_width(), accumulated\_height))

else:

raise TextRectException, "Invalid justification argument: " + str(justification)

accumulated\_height += font.size(line)[1]

return surface

#Class for levels-b(bowl),s(star),r(ring/bubble),c(centre of rotation 1),c2(centre of rotation #2),r(radius of swing)

class levels(object):

def \_\_init\_\_(self,l,xb,yb,xs1,ys1,xs2,ys2,xs3,ys3,xr,yr,xh11,yh11,xh12,yh12,d11,d12,xh21,yh21,xh22,yh22,d21,d22,xc,yc,r,xc2=0,yc2=0,r2=0):

self.level = l

self.xb = xb

self.yb = yb

self.xs1 = xs1

self.ys1 = ys1

self.xs2 = xs2

self.ys2 = ys2

self.xs3 = xs3

self.ys3 = ys3

self.r = r

self.xc = xc

self.yc = yc

self.r2 = r2

self.xc2 = xc2

self.yc2 = yc2

self.xr = xr

self.yr = yr

self.xh11 = xh11

self.yh11 = yh11

self.xh12 = xh12

self.yh12 = yh12

self.d11 = d11

self.d12 = d12

self.xh21 = xh21

self.yh21 = yh21

self.xh22 = xh22

self.yh22 = yh22

self.d21 = d21

self.d22 = d22

#function to invoke game play for respective level

def Play\_Level(self):

global s1,s2,s3,r,h1,h2,hi1,hi2

s1 = star(self.xs1,self.ys1)

s2 = star(self.xs2,self.ys2)

s3 = star(self.xs3,self.ys3)

r = ring(self.xr,self.yr)

h1 = hat(self.xh11,self.yh11,self.xh12,self.yh12,self.d11,self.d12,hat1)

hi1 = hat(self.xh12,self.yh12,self.xh11,self.yh11,self.d12,self.d11,hat1)

h2 = hat(self.xh21,self.yh21,self.xh22,self.yh22,self.d21,self.d22,hat2)

hi2 = hat(self.xh22,self.yh22,self.xh21,self.yh21,self.d22,self.d21,hat2)

if r.x==0 and r.y==0:

r.exist = False

Game\_Loop(self.level, self.xc, self.yc, self.r, self.xb, self.yb, self.xc2, self.yc2, self.r2)

#star as an object

class star:

def \_\_init\_\_(self,x,y):

self.img = stari

self.x = x

self.y = y

self.exist = True

#ring as an object

class ring:

def \_\_init\_\_(self,x,y):

self.occupied = False

self.x = x

self.y = y

self.exist = True

class hat:

def \_\_init\_\_(self,x1,y1,x2,y2,dir1,dir2,hat):

self.pos1 = (x1,y1)

self.pos2 = (x2,y2)

self.d1 = dir1 #angle wrt upwards facing,anti-clockwise

self.d2 = dir2

if x1 or y1 or x2 or y2:

self.exist = True

if self.d1 == 'r':

self.img1 = pygame.transform.rotate(hat,-90)

elif self.d1 == 'l':

self.img1 = pygame.transform.rotate(hat,90)

elif self.d1 == 'u':

self.img1 = hat

elif self.d1 == 'd':

self.img1 = pygame.transform.rotate(hat,180)

else:

print 'Not correct direction for hat!'

if self.d2 == 'r':

self.img2 = pygame.transform.rotate(hat,-90)

elif self.d2 == 'l':

self.img2 = pygame.transform.rotate(hat,90)

elif self.d2 == 'u':

self.img2 = hat

elif self.d2 == 'd':

self.img2 = pygame.transform.rotate(hat,180)

else:

print 'Not correct direction for hat!'

else:

self.exist = False

def get\_name():

c=1

font\_size = pygame.font.Font('freesansbold.ttf',int(candy\_width/2))

n=''

while c:

surface.blit(back,(0,0))

s = 'HIGHSCORE!! Please enter your name: '

high = font\_size.render(s+n, True, black)

Rect = high.get\_rect()

Rect.center = (int(display\_w/2), int(display\_h/2-candy\_width))

surface.blit(high, Rect)

pygame.display.update()

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_RETURN:

c=0

break

lst = pygame.key.get\_pressed()

for i in range(len(lst)):

if lst[i]:

n+=str(unichr(i))

break

return n

class score:

def \_\_init\_\_(self,score):

self.score = score

self.level = level

def check\_high(self):

fo = open('Highscores.DAT','rb')

L=[]

try:

L+=[pickle.load(fo)]

except:

print 'closed'

fo.close()

print L

if len(L)==1:

if L[0]<self.score:

print 2

self.name = get\_name()

L.insert(0,self)

else:

self.name = get\_name()

L.insert(1,self)

else:

for i in range(len(L)-2,-1,-1):

if L[i].level==self.level:

if L[i].score>self.score:

print 1

if L[i+1]<self.score:

print 2

self.name = get\_name()

L.insert(i+1,self)

break

else:

print 3

break

else:

print 4

self.name = get\_name()

L.insert(0,self)

if len(L)>3:

print 5

del L[-1]

fo = open('Highscores.DAT','wb')

for i in L:

print 6

pickle.dump(i,fo)

fo.close()

##initialising

l1=levels(1,550,500,280,330,350,350,400,400,0,0,0,0,0,0,'','',0,0,0,0,'','',200,200,150,0,0,0)

l1=levels(1,500,500,330,270,400,250,460,300,0,0,0,0,0,0,'','',0,0,0,0,'','',200,200,150,0,0,0)

l2=levels(2,677-candy\_width\*2,600,450,250,650,135,345,252,677,406,0,0,0,0,'','',0,0,0,0,'','',200,50,200,0,0,0)

l3=levels(3,450,500,330,270,200,100,460,300,250,500,200,350,600,400,'d','l',200,600,500,300,'u','d',200,200,150,0,0,0)

angle\_max = (math.pi/2)

check=0

level = None

#starting game

fo = open('Instructions.txt')

r = '\n\n'+fo.read()

fo.close()

font\_size = pygame.font.Font('freesansbold.ttf',30)

instr = font\_size.render(r, True, black)

##Rect = instr.get\_rect()

##Rect.center = (int(display\_w/2), int(display\_h/2-candy\_width\*3))

surface.blit(back1,(0,0))

Rect = pygame.Rect((25, 25, display\_w-50, display\_h-50))

rendered\_text = render\_textrect(r, font\_size, Rect, (216, 216, 216), (48, 48, 48), 1)

if rendered\_text:

surface.blit(rendered\_text, Rect.topleft)

pygame.display.update()

c=1

while c:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

quit()

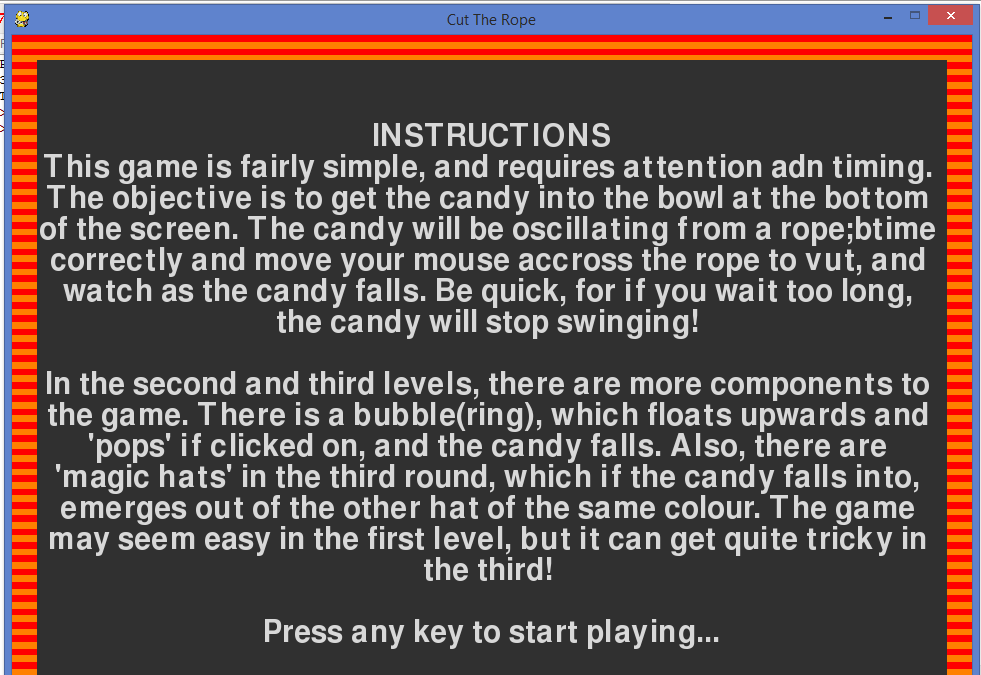
if event.type == pygame.KEYUP:

c=0

Menu()

*Screen shots of outputs :*

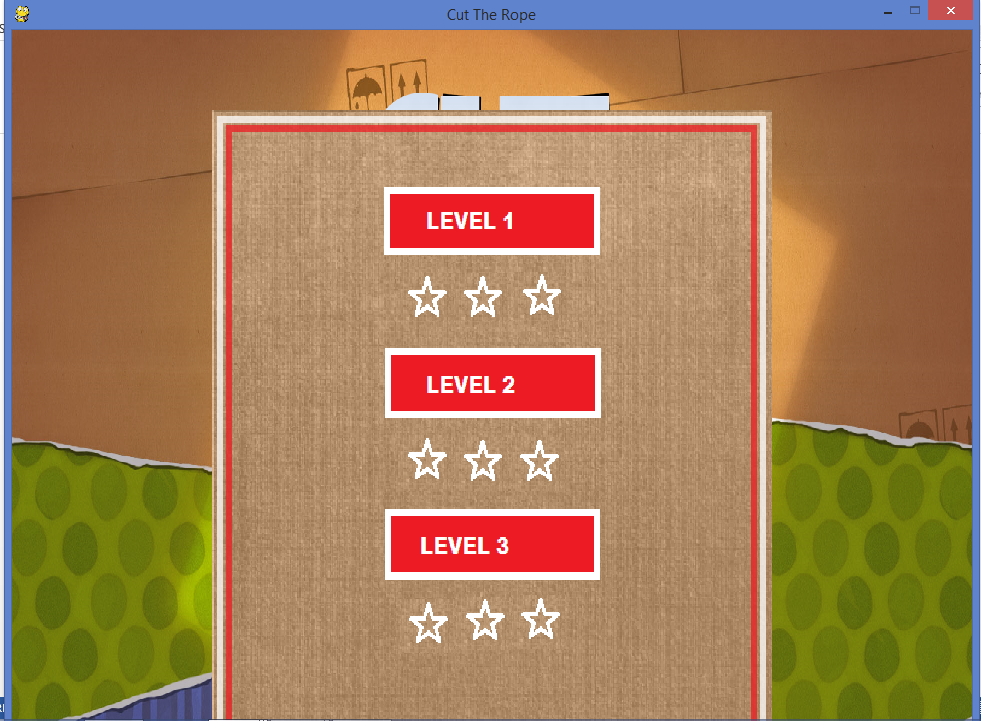
# initial instructions to the player



# menu selection option



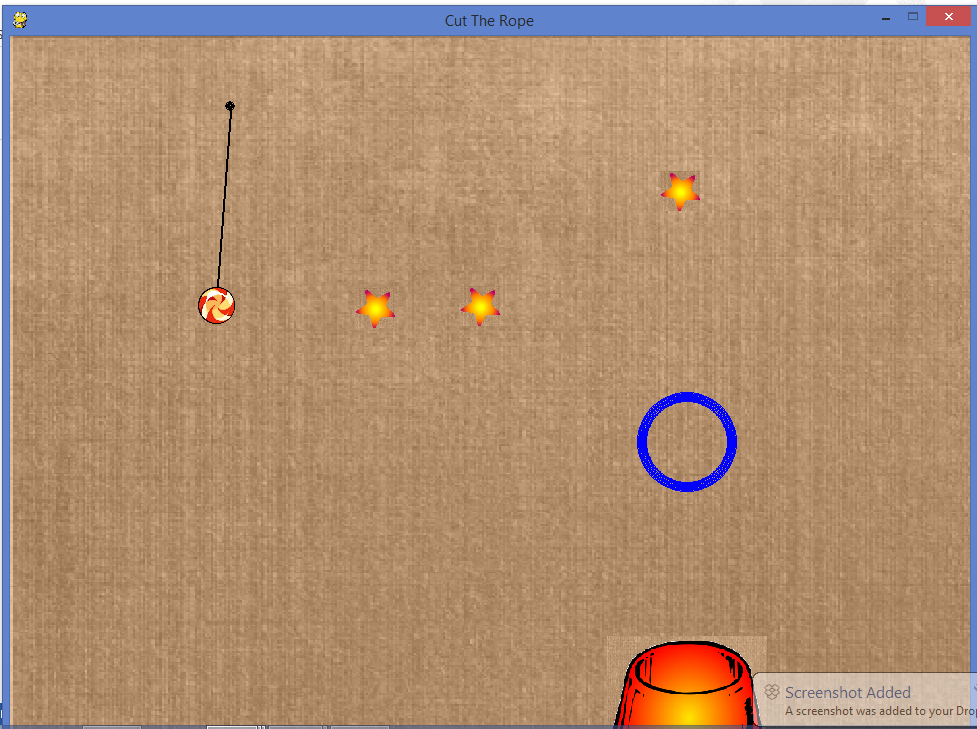
# level selection option



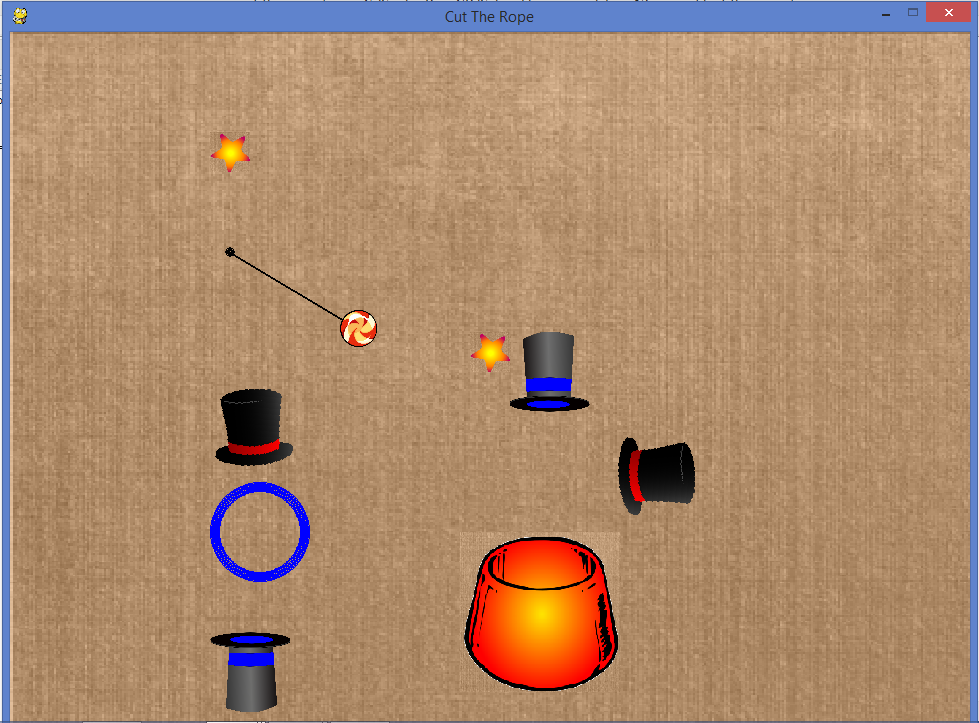
# level 1



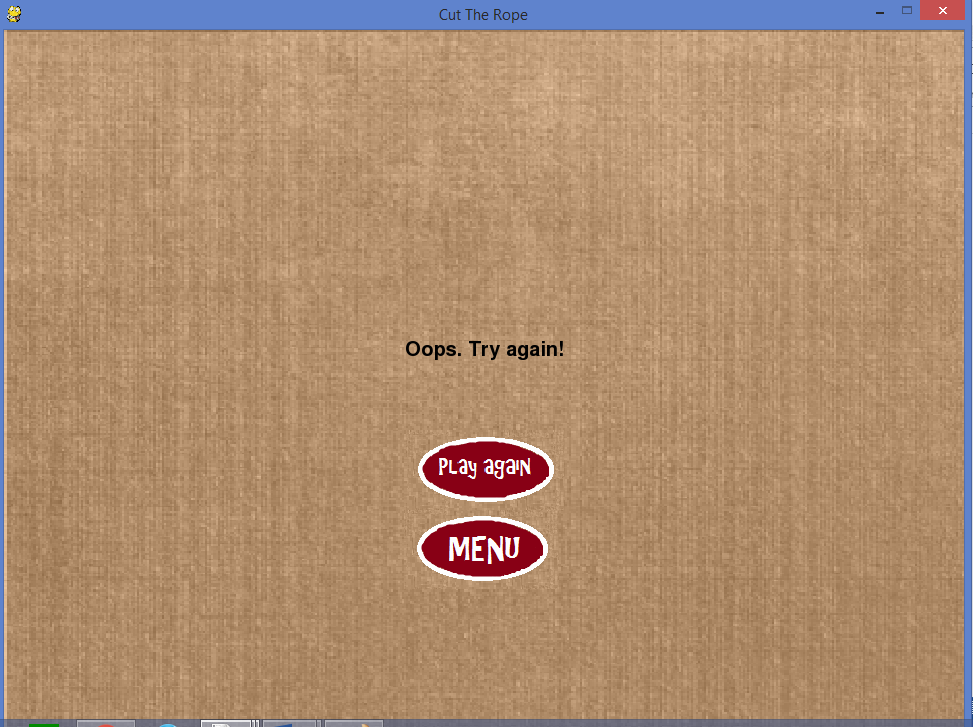
# level 2



# level 3



# if game lost



*Bibliography :*

* [www.**pygame**.org/](http://www.pygame.org/)
* [**www.stackoverflow**.com/](http://www.stackoverflow.com/)
* https://pythonprogramming.net/**pygame**-python-3-part-1-intro/
* https://www.youtube.com/watch