A MINI PROJECT

on

Ratatouille

(A Python Game)

(Submitted for partial fulfilment of the requirements for the award of the degree)

of

BE 2/4(CSE) MINI PROJECT

BY

SAMALA SUSHANTH

(160115733115)

under the guidance of

Mr.B.RAMADASU

Assistant Professor

Department of IT

CBIT, Hyderabad

Department of Computer Science and Engineering

Chaitanya Bharathi Institute of Technology

Gandipet, Hyderabad-500075

CERTIFICATE

This is to certify that the project work entitled "**Ratatouille**" is bonafide work carried out by **S.Sushanth(160115733115)** in fulfilment of the award of the degree of **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING** by the **OSMANIA UNIVERSITY**, Hyderabad, under our guidance and supervision.

The results enclosed in this report are not submitted to any other university or organisation for the award of any degree or diploma.

Internal guide	Head of the department
Mr.B.RAMADASU	Dr.Y.RamaDevi
Assistant professor	Professor and Head
Department of IT	Department of CSE
CBIT, Hyderabad	CBIT, Hyderabad

DECLARATION

This to certify that the work reported in the present project entitled **"Ratatouille"** is a record of work done by us in the Department of Computer Science and Engineering, Chaitanya Bharathi Institute of Technology, Osmania University . The reports are based on the project work done entirely by us and not copied from any source.

S . S U S H A N T H
(160115733115)

ACKNOWLEDGEMENTS

We would like to express my deep felt appreciation and gratitude to **Mr.B.RAMADASU,** Assistant Professor,Department of CSE, my project guide, for her skilful guidance , constant supervision, timely suggestion ,keen interest and encouragement in completing the project within the stipulated time.

We are honoured to express our profound sense of gratitude to **Dr.Y.Rama Devi,** Head of Department, CSE, who has served as a host of valuable corrections and for providing us time and amenities to complete this project.

We gratefully express our thanks to **Dr.ChennaKeshava Rao,** Principal of our college and the management of **"CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY"** for providing excellent academic and learning environment in the college.

We wish to express our heartfelt gratitude to the **Members of staff** and all others who helped us in bringing up out project. We would also like to thank the **Lab assistants and programmers** for helping us through our project.

ABSTRACT

This project involves a game in which basically a bunny soldier is given the duty to protect the castle from the evaders within the lifespan of 1minute and 30secondsThis project empowers the use of python in the gaming universe, this game is the measure of how efficient and convenient the python language is for programmers.

The game is built upon the python library named PyGame, PyAudio, PyVideo, it contains a touch of genius which calculates the accuracy of your performance in the game.The game Ratatouille is a very flexible game and can be modified without any hassle.

TABLE OF CONTENTS

Chapter 1
INTRODUCTION

1.1 Introduction to Ratatouille

Ratatouille is a Python game in which a bunny soldier is set on to protect the castle and the king from the evaders from another kingdom, he can move in four directions and shoot bullets in all the possible directions and angles and is supposed to prevent the enemies from entering the castle before the given stipulated time which is 1minute 30seconds , as the enemies approach the castle the health bar at the left top corner of the screen decreases and the game is over either when the time runs out or the health bar goes to zero.

Pygame offers control over the system hardware cursor. Pygame only supports black and white cursors for the system. You control the cursor with functions inside pygame.mouse.

This cursors module contains functions for loading and unencoding various cursor formats. These allow you to easily store your cursors in external files or directly as encoded python strings.

The module includes several standard cursors. The pygame.mouse.set_cursor() function takes several argu- ments.

Chapter 2

REQUIREMENTS

3.1. Hardware Requirements

Processor	: Intel Pentium or equivalent
RAM	: 2GB
Hard disk	: 40 GB or more

3.2 Software Requirements

Programming Language	: Python
Operating System	: Windows (7, 8, 8.1 or 10) , Linux, Unix

Chapter 3
DESIGN OF PROJECT

4.1Module Description

pygame (**the library**) is a Free and Open Source python programming language library for making multimedia applications like games built on top of the excellent SDL library. Like SDL, pygame is highly portable and runs on nearly every platform and operating system. Millions of people have downloaded pygame itself, which is a whole lot of bits flying across the interwebs.
pygame.org (**the website**) welcomes all Python game, art, music, sound, video and multimedia projects. Once you have finished getting started you could add a new project or learn about pygame by reading the docs. For more information on what is happening in the pygame world see the community dashboard, which lists many things like our projects we are working on, news (our blog with rss), twitter, reddit (forum), stackoverflow (Q&A), Bitbucket (development), irc (chat), mailinglist (we love writing electronic mail to each other) and other various bits and pieces about pygame from around the internets.

Ratatouille is made with the help of PyGame module and has a unique story and graphics

class pygame.Color

pygame object for color representations Color(name) -> Color

Color(r, g, b, a) -> Color Color(rgbvalue) -> Color

The Color class represents RGBA color values using a value range of 0-255. It allows basic arithmetic operations to create new colors, supports conversions to other color spaces such as HSV or HSL and lets you adjust single color channels. Alpha defaults to 255 when not given.

‘rgbvalue’ can be either a color name, an HTML color format string, a hex number string, or an integer pixel value. The HTML format is ‘#rrggbbaa’, where rr, gg, bb, and aa are 2 digit hex numbers. The alpha aa is optional. A hex number string has the form ‘0xrrggbbaa’, where aa is optional.

Color objects support equality comparison with other color objects and 3 or 4 element tuples of integers (New in 1.9.0). There was a bug in pygame 1.8.1 where the default alpha was 0, not 255 like previously.

New implementation of Color was done in pygame 1.8.1.

Gets or sets the red value of the Color. r -> int

The red value of the Color.

Gets or sets the green value of the Color. g -> int

The green value of the Color.

Gets or sets the blue value of the Color. b -> int

The blue value of the Color.

a

cmy

hsva

Gets or sets the HSVA representation of the Color. hsva -> tuple

The HSVA representation of the Color. The HSVA components are in the ranges H = [0, 360], S = [0, 100], V = [0, 100], A = [0, 100]. Note that this will not return the absolutely exact HSV values for the set RGB values in all cases. Due to the RGB mapping from 0-255 and the HSV mapping from 0-100 and 0-360 rounding errors may cause the HSV values to differ slightly from what you might expect.

hsla

Gets or sets the HSLA representation of the Color. hsla -> tuple

The HSLA representation of the Color. The HSLA components are in the ranges H = [0, 360], S = [0, 100], V = [0, 100], A = [0, 100]. Note that this will not return the absolutely exact HSL values for the set RGB values in all cases. Due to the RGB mapping from 0-255 and the HSL mapping from 0-100 and 0-360 rounding errors may cause the HSL values to differ slightly from what you might expect.

i1i2i3

Gets or sets the I1I2I3 representation of the Color. i1i2i3 -> tuple

The I1I2I3 representation of the Color. The I1I2I3 components are in the ranges I1 = [0, 1], I2 = [-0.5, 0.5], I3 = [-0.5, 0.5]. Note that this will not return the absolutely exact I1I2I3 values for the set RGB values in all cases. Due to the RGB mapping from 0-255 and the I1I2I3 mapping from 0-1 rounding errors may cause the I1I2I3 values to differ slightly from what you might expect.

normalize()

Returns the normalized RGBA values of the Color.

normalize() -> tuple

Returns the normalized RGBA values of the Color as floating point values.
correct_gamma()

Gets or sets the alpha value of the Color. a -> int

The alpha value of the Color.

Gets or sets the CMY representation of the Color. cmy -> tuple

The CMY representation of the Color. The CMY components are in the ranges C = [0, 1], M = [0, 1], Y = [0, 1]. Note that this will not return the absolutely exact CMY values for the set RGB values in all cases. Due to the RGB mapping from 0-255 and the CMY mapping from 0-1 rounding errors may cause the CMY values to differ slightly from what you might expect.

Applies a certain gamma value to the Color. correct_gamma (gamma) -> Color

Applies a certain gamma value to the Color and returns a new Color with the adjusted RGBA values. set_length()

Set the number of elements in the Color to 1,2,3, or 4. set_length(len) -> None

The default Color length is 4. Colors can have lengths 1,2,3 or 4. This is useful if you want to unpack to r,g,b and not r,g,b,a. If you want to get the length of a Color do len(acolor).

New in pygame 1.9.0.

Chapter 4

IMPLEMENTATION

4.1 Ratatouille.py

```
####################################
#          4th Semester Project    #
#                 By               #
#            Sushanth Samala       #
#          Roll No.160115733115    #
####################################

# 1 - Import library
import pygame
from pygame.locals import *
import math
import random

# 2 - Initialize the game
pygame.init()
width, height = 640, 480
screen=pygame.display.set_mode((width, height))
keys = [False, False, False, False]
playerpos=[100,100]
acc=[0,0]
arrows=[]
badtimer=100
badtimer1=0
badguys=[[640,100]]
healthvalue=194
pygame.mixer.init()

# 3 - Load image
player = pygame.image.load("resources/images/dude.png")
grass = pygame.image.load("resources/images/grass.png")
castle = pygame.image.load("resources/images/castle.png")
arrow = pygame.image.load("resources/images/bullet.png")
badguyimg1 = pygame.image.load("resources/images/badguy.png")
badguyimg=badguyimg1
healthbar = pygame.image.load("resources/images/healthbar.png")
```

```
health = pygame.image.load("resources/images/health.png")
gameover = pygame.image.load("resources/images/gameover.png")
youwin = pygame.image.load("resources/images/youwin.png")
# 3.1 - Load audio
hit = pygame.mixer.Sound("resources/audio/explode.wav")
enemy = pygame.mixer.Sound("resources/audio/enemy.wav")
shoot = pygame.mixer.Sound("resources/audio/shoot.wav")
hit.set_volume(0.05)
enemy.set_volume(0.05)
shoot.set_volume(0.05)
pygame.mixer.music.load('resources/audio/moonlight.wav')
pygame.mixer.music.play(-1, 0.0)
pygame.mixer.music.set_volume(0.25)

# 4 - keep looping through
running = 1
exitcode = 0
while running:
    badtimer-=1
    # 5 - clear the screen before drawing it again
    screen.fill(0)
    # 6 - draw the player on the screen at X:100, Y:100
    for x in range(width/grass.get_width()+1):
        for y in range(height/grass.get_height()+1):
            screen.blit(grass,(x*100,y*100))
    screen.blit(castle,(0,30))
    screen.blit(castle,(0,135))
    screen.blit(castle,(0,240))
    screen.blit(castle,(0,345 ))
    # 6.1 - Set player position and rotation
    position = pygame.mouse.get_pos()
            angle = math.atan2(position[1]-(playerpos[1]+32),position[0]-
(playerpos[0]+26))
    playerrot = pygame.transform.rotate(player, 360-angle*57.29)
            playerpos1 = (playerpos[0]-playerrot.get_rect().width/2, playerpos[1]-
playerrot.get_rect().height/2)
    screen.blit(playerrot, playerpos1)
    # 6.2 - Draw arrows
    for bullet in arrows:
        index=0
```

```
    velx=math.cos(bullet[0])*10
    vely=math.sin(bullet[0])*10
    bullet[1]+=velx
    bullet[2]+=vely
    if bullet[1]<-64 or bullet[1]>640 or bullet[2]<-64 or bullet[2]>480:
        arrows.pop(index)
    index+=1
    for projectile in arrows:
        arrow1 = pygame.transform.rotate(arrow, 360-projectile[0]*57.29)
        screen.blit(arrow1, (projectile[1], projectile[2]))
# 6.3 - Draw badgers
if badtimer==0:
    badguys.append([640, random.randint(50,430)])
    badtimer=100-(badtimer1*2)
    if badtimer1>=35:
        badtimer1=35
    else:
        badtimer1+=5
index=0
for badguy in badguys:
    if badguy[0]<-64:
        badguys.pop(index)
    badguy[0]-=7
    # 6.3.1 - Attack castle
    badrect=pygame.Rect(badguyimg.get_rect())
    badrect.top=badguy[1]
    badrect.left=badguy[0]
    if badrect.left<64:
        hit.play()
        healthvalue -= random.randint(5,20)
        badguys.pop(index)
    #6.3.2 - Check for collisions
    index1=0
    for bullet in arrows:
        bullrect=pygame.Rect(arrow.get_rect())
        bullrect.left=bullet[1]
        bullrect.top=bullet[2]
        if badrect.colliderect(bullrect):
            enemy.play()
            acc[0]+=1
```

```
                badguys.pop(index)
                arrows.pop(index1)
            index1+=1
        # 6.3.3 - Next bad guy
        index+=1
    for badguy in badguys:
        screen.blit(badguyimg, badguy)
    # 6.4 - Draw clock
    font = pygame.font.Font(None, 24)
                survivedtext = font.render(str((90000-pygame.time.get_ticks())/
60000)+":"+str((90000-pygame.time.get_ticks())/1000%60).zfill(2), True, (0,0,0))
    textRect = survivedtext.get_rect()
    textRect.topright=[635,5]
    screen.blit(survivedtext, textRect)
    # 6.5 - Draw health bar
    screen.blit(healthbar, (5,5))
    for health1 in range(healthvalue):
        screen.blit(health, (health1+8,8))
    # 7 - update the screen
    pygame.display.flip()
    # 8 - loop through the events
    for event in pygame.event.get():
        # check if the event is the X button
        if event.type==pygame.QUIT:
            # if it is quit the game
            pygame.quit()
            exit(0)
        if event.type == pygame.KEYDOWN:
            if event.key==K_w:
                keys[0]=True
            elif event.key==K_a:
                keys[1]=True
            elif event.key==K_s:
                keys[2]=True
            elif event.key==K_d:
                keys[3]=True
        if event.type == pygame.KEYUP:
            if event.key==pygame.K_w:
                keys[0]=False
            elif event.key==pygame.K_a:
```

```
            keys[1]=False
        elif event.key==pygame.K_s:
            keys[2]=False
        elif event.key==pygame.K_d:
            keys[3]=False
    if event.type==pygame.MOUSEBUTTONDOWN:
        shoot.play()
        position=pygame.mouse.get_pos()
        acc[1]+=1
            arrows.append([math.atan2(position[1]-(playerpos1[1]+32),position[0]-
(playerpos1[0]+26)),playerpos1[0]+32,playerpos1[1]+32])

    # 9 - Move player
    if keys[0]:
        playerpos[1]-=5
    elif keys[2]:
        playerpos[1]+=5
    if keys[1]:
        playerpos[0]-=5
    elif keys[3]:
        playerpos[0]+=5

    #10 - Win/Lose check
    if pygame.time.get_ticks()>=90000:
        running=0
        exitcode=1
    if healthvalue<=0:
        running=0
        exitcode=0
    if acc[1]!=0:
        accuracy=acc[0]*1.0/acc[1]*100
    else:
        accuracy=0

# 11 - Win/lose display
if exitcode==0:
    pygame.font.init()
    font = pygame.font.Font(None, 24)
    text = font.render("Accuracy: "+str(accuracy)+"%", True, (255,0,0))
    textRect = text.get_rect()
```

```
    textRect.centerx = screen.get_rect().centerx
    textRect.centery = screen.get_rect().centery+24
    screen.blit(gameover, (0,0))
    screen.blit(text, textRect)
else:
    pygame.font.init()
    font = pygame.font.Font(None, 24)
    text = font.render("Accuracy: "+str(accuracy)+"%", True, (0,255,0))
    textRect = text.get_rect()
    textRect.centerx = screen.get_rect().centerx
    textRect.centery = screen.get_rect().centery+24
    screen.blit(youwin, (0,0))
    screen.blit(text, textRect)
while 1:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            exit(0)
    pygame.display.flip()
```

4.2 Resources

Images:

Badguy.png

Badguy1.png

Badguy2.png

Badguy3.png

Badguy4.png

Bullet.png

Castle.png

Dude.png

Dude2.png

Health.png

Healthcare.png

Grass.png

Youwin.png

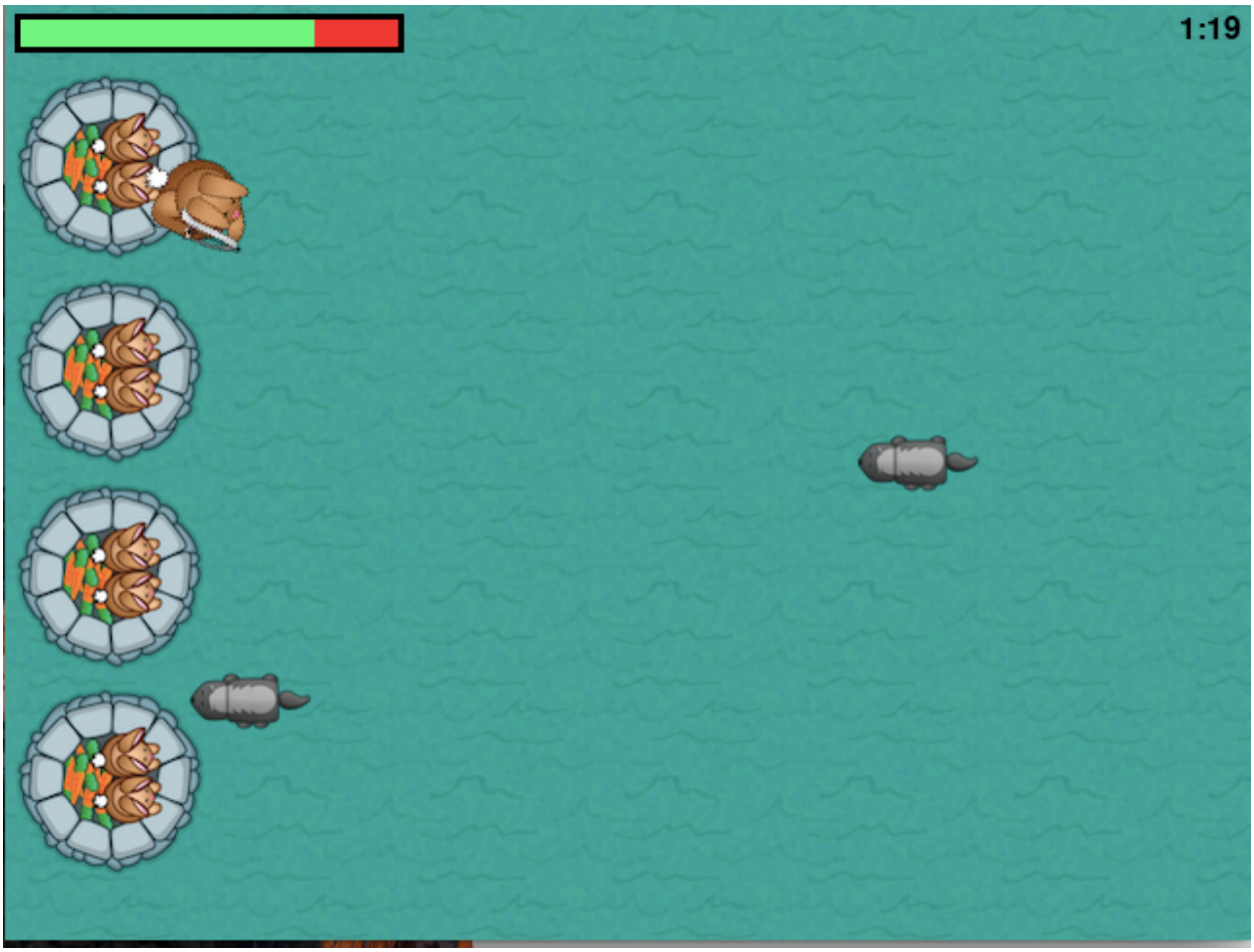
Gameover.png

Audio:

explode.wav
moonlight.wav
shoot.wav
enemy.wav

Chapter 5

RESULTS AND DISCUSSIONS



Homepage: This is the screenshot of the game in progress

End: Screenshot when the game is over

Chapter 6

CONCLUSION AND FURTHER ENHANCEMENT

This program is designed in such a manner that future changes can be made as long as the programmer accounts for all of changes which must be made in the relevant files. For example, if an additional field is added to the images or sound, then the new files can be added to the images directory and the audio directory respectively. The visuals can easily be changed through the usage of PyGame module functions. The following conclusions can be deduced from the development of the project:

- The game is user friendly. It is easy to understand.
- It can be easily modified according to the programmer's need.
- This project is a very good experience for us. We learned about the Python audio and graphics modules.
- We believe that this project has a lot of potential growth because many changes can be made to enhance it however we were unable to implement them due to time constraints.
- We can use PyAudio improve the audio of the game.
- We can add the functionality of remembering the previous score.
- We can host it on the Internet, Chrome, Firefox, etc. using a web hosting service. This will allow anyone to view the game even if they don't have the source code and XAMPP.

Chapter 7

REFERENCES

1. https://kivy.org/docs/tutorials/pong.html
2. https://media.readthedocs.org/pdf/pygame/latest/pygame.pdf
3. https://docs.blender.org/api/2.78b/
4. http://www.pygame.org
5. https://wiki.python.org/moin/GameProgramming
6. https://inventwithpython.com