

**MINI PROJECT USING OPEN TECHNOLOGY**

**(ACSE-0459)**

**Affiliated to Dr. A.P.J Abdul Kalam Technical University, Uttar Pradesh**

**Ms. RITIKA & Ms. SURBHI PURWA Name: Garv Kumar**

**Roll No: 2101330109003**

**Submitted To: Submitted By:**

**(IV-SEMSTER)**

**LAB FILE**

**ON**

**Session (2021 – 2022)**

COMPUTER SCIENCE & ENGINEERING

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY

GREATER NOIDA-201306

(An Autonomous Institute)

College of Computer Science & Engineering

Project Report

on

**FLAPPY BIRD**

By

GARV KUMAR (2101330109003) CSE IV SEM D2

Under the Supervision of

Ms. RITIKA & Ms. SURBHI PURWA

(COORDINATORS)

Submitted to the department of Computer Science and Engineering

For the partial fulfillment of the requirements

for award of Bachelor of Technology

in

Computer Science and Engineering



**Noida Institute of Engineering & Technology Gr. Noida**

**Dr. A.P.J. Abdul Kalam Technical University, Lucknow, Uttar Pradesh, India**

**May, 2021-2022**

**Certificate­­**

### This is to certify that the Project report entitled “FLAPPY BIRD USING PYTHON(PYGAME)” is a record of the work done by the following students:

### Student name Roll No.

GARV KUMAR 2101330109003

### This work is done under my/our supervision and guidance during the academic year of 2021-22.This report is submitted to the Noida Institute of Engineering & Technology, Greater Noida for partial fulfillment for the degree of B.TECH. (Computer Science and Engineering) of Dr A P J Abdul Kalam Technical University, Lucknow, Uttar Pradesh, India.

I/We wish him/her all the best for all the endeavors.

Signature of Guide:

Ms. RITIKA & Ms. SURBHI PURWA (ASSISTANT PROFESSOR CSE)

**ACKNOWLEDGEMENT**

I would like to place on record my deep sense of gratitude to Ms. RITIKA & Ms. SURBHI PURWA ASSISTANT PROFESSOR **Department of Computer Science and Engineering, Noida Institute of Engineering & Technology**, Greater Noida, Gautam Budha Nagar, Uttar Pradesh, India For his generous guidance, help and useful suggestions.

I express my sincere gratitude to **Prof. Chandra Shekhar Yadav , HODCSE**, Noida Institute of Engineering & Technology, Greater Noida for his stimulating guidance, continuous encouragement and supervision throughout the course of present work.

I express my sincere gratitude to **MY COLLEAGUES FACULTY** Noida Institute of Engineering & Technology, Greater Noida for his stimulating guidance, continuous encouragement and supervision throughout the course of present work.

**Date: Student Name:**

GARV KUMAR

**ABSTRACT**

Flappy Bird is an endless game that involves a bird that the player can control. The player has to save the bird from colliding with the hurdles like pipes. Each time the bird passes through the pipes, the score gets incremented by one. The game ends when the bird collides with the pipes or falls down due to gravity. The sections below describe the steps that have to be taken for building this game.

The aim of this paper is to develop and study an artiﬁcial intelligence

based game-playing agent using genetic algorithm and neural networks. We ﬁrst

create an agent which learns how to optimally play the famous “Flappy Bird”

game by safely dodging all the barriers and ﬂapping its way through them and

then study the effect of changing various parameters like number of neurons on

the hidden layer, gravity, speed, gap between trees has on the learning process.

The gameplay was divided into two level of difﬁculty to facilitate study on the

learning process. Phaser Framework was used to facilitate HTML5 program-

ming for introducing real-life factors like gravity, collision and Synaptic Neural

Network library was used to implement neural network so as to avoid creating a

neural network from scratch. Machine Learning Algorithm which we have

adopted in this project is based on the concept of Neuro-Evolution and this form

of machine learning uses algorithms which can evolve and mature over time

such as a genetic algorithm to train artiﬁcial neural networks.

Keywords: Artiﬁcial Intelligence Neural network Genetic algorithm AI 

Game-playing agent Flappy bird

The aim of this paper is to develop and study an artiﬁcial intelligence

based game-playing agent using genetic algorithm and neural networks. We ﬁrst

create an agent which learns how to optimally play the famous “Flappy Bird”

game by safely dodging all the barriers and ﬂapping its way through them and

then study the effect of changing various parameters like number of neurons on

the hidden layer, gravity, speed, gap between trees has on the learning process.

The gameplay was divided into two level of difﬁculty to facilitate study on the

learning process. Phaser Framework was used to facilitate HTML5 program-

ming for introducing real-life factors like gravity, collision and Synaptic Neural

Network library was used to implement neural network so as to avoid creating a

neural network from scratch. Machine Learning Algorithm which we have

adopted in this project is based on the concept of Neuro-Evolution and this form

of machine learning uses algorithms which can evolve and mature over time

such as a genetic algorithm to train artiﬁcial neural networks.

Keywords: Artiﬁcial Intelligence Neural network Genetic algorithm AI 

Game-playing agent Flappy bird

The aim of this paper is to develop and study an artiﬁcial intelligence

based game-playing agent using genetic algorithm and neural networks. We ﬁrst

create an agent which learns how to optimally play the famous “Flappy Bird”

game by safely dodging all the barriers and ﬂapping its way through them and

then study the effect of changing various parameters like number of neurons on

the hidden layer, gravity, speed, gap between trees has on the learning process.

The gameplay was divided into two level of difﬁculty to facilitate study on the

learning process. Phaser Framework was used to facilitate HTML5 program-

ming for introducing real-life factors like gravity, collision and Synaptic Neural

Network library was used to implement neural network so as to avoid creating a

neural network from scratch. Machine Learning Algorithm which we have

adopted in this project is based on the concept of Neuro-Evolution and this form

of machine learning uses algorithms which can evolve and mature over time

such as a genetic algorithm to train artiﬁcial neural networks.

Keywords: Artiﬁcial Intelligence Neural network Genetic algorithm AI 

Game-playing agent Flappy bird

The aim of this paper is to develop and study an artiﬁcial intelligence

based game-playing agent using genetic algorithm and neural networks. We ﬁrst

create an agent which learns how to optimally play the famous “Flappy Bird”

game by safely dodging all the barriers and ﬂapping its way through them and

then study the effect of changing various parameters like number of neurons on

the hidden layer, gravity, speed, gap between trees has on the learning process.

The gameplay was divided into two level of difﬁculty to facilitate study on the

learning process. Phaser Framework was used to facilitate HTML5 program-

ming for introducing real-life factors like gravity, collision and Synaptic Neural

Network library was used to implement neural network so as to avoid creating a

neural network from scratch. Machine Learning Algorithm which we have

adopted in this project is based on the concept of Neuro-Evolution and this form

of machine learning uses algorithms which can evolve and mature over time

such as a genetic algorithm to train artiﬁcial neural networks.

Keywords: Artiﬁcial Intelligence Neural network Genetic algorithm AI 

Game-playing agent Flappy bird

The aim of this paper is to develop and study an artiﬁcial intelligence

based game-playing agent using genetic algorithm and neural networks. We ﬁrst

create an agent which learns how to optimally play the famous “Flappy Bird”

game by safely dodging all the barriers and ﬂapping its way through them and

then study the effect of changing various parameters like number of neurons on

the hidden layer, gravity, speed, gap between trees has on the learning process.

The gameplay was divided into two level of difﬁculty to facilitate study on the

learning process. Phaser Framework was used to facilitate HTML5 program-

ming for introducing real-life factors like gravity, collision and Synaptic Neural

Network library was used to implement neural network so as to avoid creating a

neural network from scratch. Machine Learning Algorithm which we have

adopted in this project is based on the concept of Neuro-Evolution and this form

of machine learning uses algorithms which can evolve and mature over time

such as a genetic algorithm to train artiﬁcial neural networks.

Keywords: Artiﬁcial Intelligence Neural network Genetic algorithm AI 

Game-playing agent Flappy bird

The aim of this paper is to develop and study an artiﬁcial intelligence

based game-playing agent using genetic algorithm and neural networks. We ﬁrst

create an agent which learns how to optimally play the famous “Flappy Bird”

game by safely dodging all the barriers and ﬂapping its way through them and

then study the effect of changing various parameters like number of neurons on

the hidden layer, gravity, speed, gap between trees has on the learning process.

The gameplay was divided into two level of difﬁculty to facilitate study on the

learning process. Phaser Framework was used to facilitate HTML5 program-

ming for introducing real-life factors like gravity, collision and Synaptic Neural

Network library was used to implement neural network so as to avoid creating a

neural network from scratch. Machine Learning Algorithm which we have

adopted in this project is based on the concept of Neuro-Evolution and this form

of machine learning uses algorithms which can evolve and mature over time

such as a genetic algorithm to train artiﬁcial neural networks.

Keywords: Artiﬁcial Intelligence Neural network Genetic algorithm AI 

Game-playing agent Flappy bird

The aim of this paper is to develop and study an artiﬁcial intelligence

based game-playing agent using genetic algorithm and neural networks. We ﬁrst

create an agent which learns how to optimally play the famous “Flappy Bird”

game by safely dodging all the barriers and ﬂapping its way through them and

then study the effect of changing various parameters like number of neurons on

the hidden layer, gravity, speed, gap between trees has on the learning process.

The gameplay was divided into two level of difﬁculty to facilitate study on the

learning process. Phaser Framework was used to facilitate HTML5 program-

ming for introducing real-life factors like gravity, collision and Synaptic Neural

Network library was used to implement neural network so as to avoid creating a

neural network from scratch. Machine Learning Algorithm which we have

adopted in this project is based on the concept of Neuro-Evolution and this form

of machine learning uses algorithms which can evolve and mature over time

such as a genetic algorithm to train artiﬁcial neural networks.

Keywords: Artiﬁcial Intelligence Neural network Genetic algorithm AI 

Game-playing agent Flappy bird

The aim of this paper is to develop and study an artificial intelligence based game-playing agent using genetic algorithm and neural networks. We first create an agent which learns how to optimally play the famous “Flappy Bird” game by safely dodging all the barriers and flapping its way through them and then study the effect of changing various parameters like number of neurons on the hidden layer, gravity, speed, gap between trees has on the learning process. The gameplay was divided into two level of difficulty to facilitate study on the learning process. Phaser Framework was used to facilitate HTML5 programming for introducing real-life factors like gravity, collision and Synaptic Neural Network library was used to implement neural network so as to avoid creating a neural network from scratch. Machine Learning Algorithm which we have adopted in this project is based on the concept of Neuro-Evolution and this form of machine learning uses algorithms which can evolve and mature over time such as a genetic algorithm to train artificial neural networks.

Keywords: Artificial Intelligence Neural network Genetic algorithm AI Game-playing agent Flappy bird PYTHON PYGAME

**TABLE OF CONTENTS**

**Page No.**

**Certificate i**

**Acknowledgement ii**

**Abstracts iii**

**Table of Contents iv**

**Table of Contents**

1. Overview......................................................................................... 5

2. High Level Design ............................................................................ 6

3. Game Logic Controller..................................................................... 7

4. Game stuff preparation .................................................................. 8

5. VGA Device Drive ............................................................................ 9

6. Sprite Controllers and VGA Display................................................. 9

7. Audio............................................................................................... 10

8.. Source Code .............................................................................. 14

**1. Overview**

In this project, we design and implement a Flappy Bird like video game on the PYGame development framework. Flappy Bird is a very popular mobile game on Android platform, driving a lot of people crazy. In this game, the player can control the vertical movement of bird ( every pressing on the keyboard makes the bird leap upward for a little bit, and the bird will fall freely without control ). As soon as the game begins, tubes will keep appearing from the right side of the screen and moving leftwards. (so that it seems like the bird flying forward). The goal of this game is to control the bird, dodging and passing the incoming tubes, as many as possible. The game is endless until the bird eventually hit one of the tubes, ground, or ceiling. Figure 1 is the start screen of Flappy Bird. The title "Flappy Bird" is shown in the middle of the uppers side of the screen. The bird is also displayed on the background.

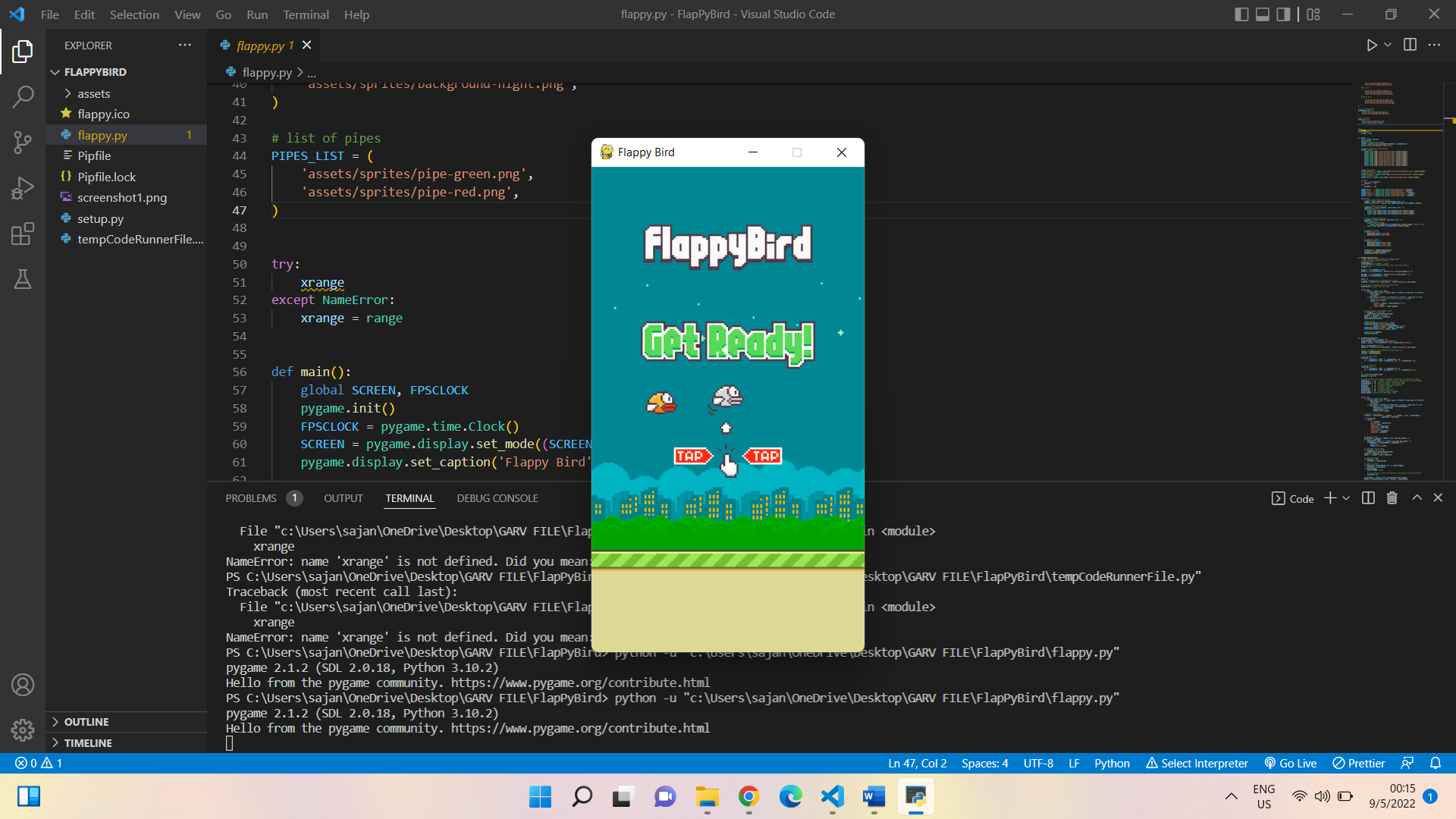


Figure 1. Start screen for Flappy Bird

Figure 2 shows the screen when the game is on. The many pillars are displayed on the screen, and so is the score, on top of the background or the pillar. (instead of the title)

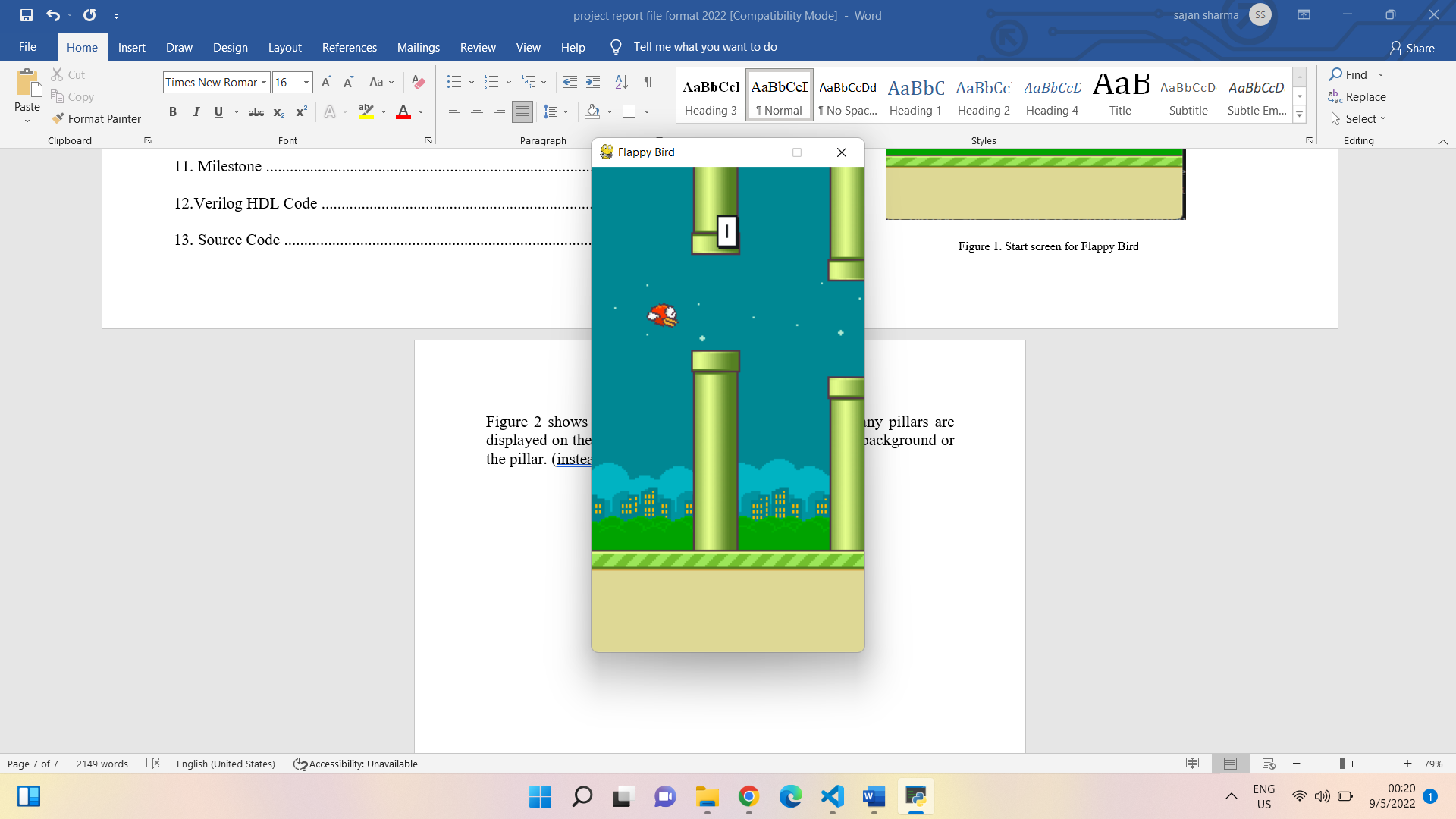
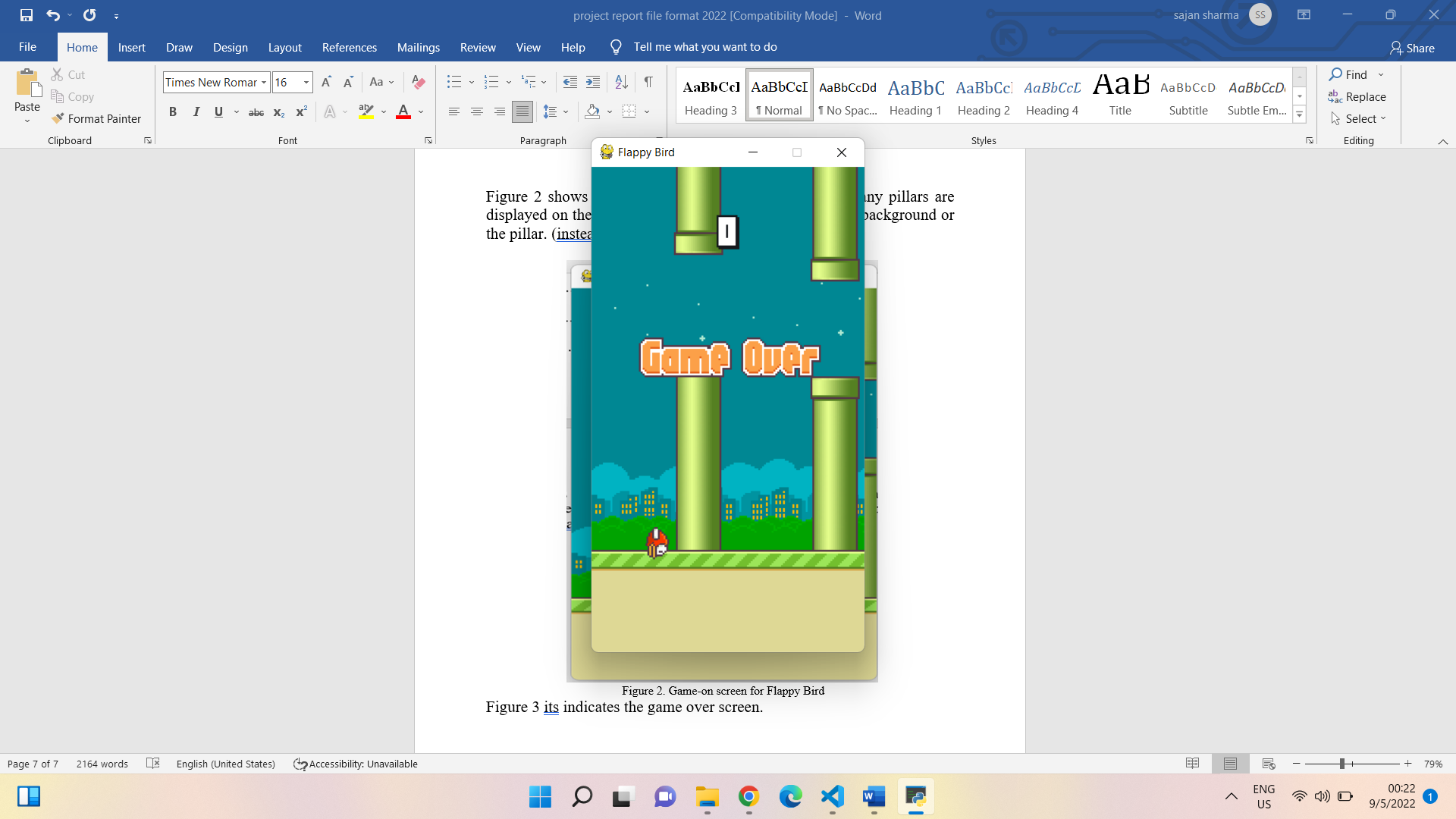


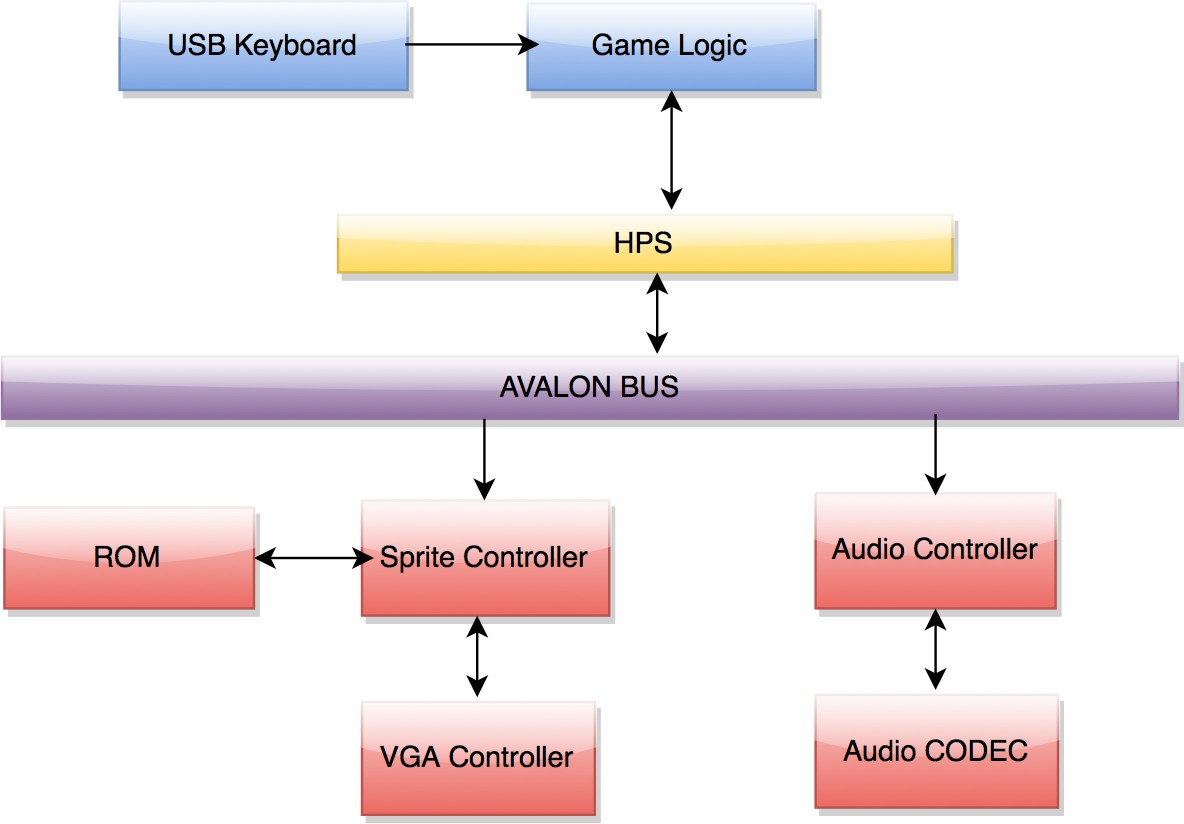
Figure 2. Game-on screen for Flappy Bird

Figure 3 its indicates the game over screen.



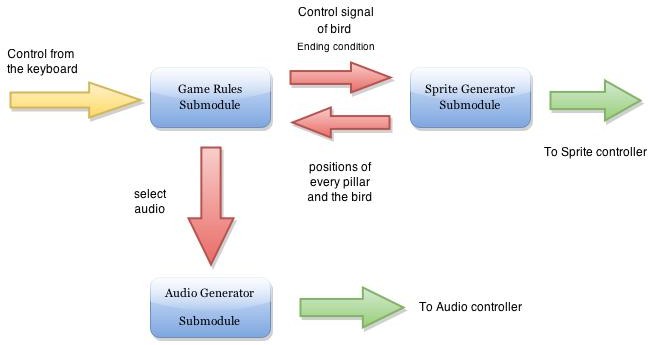
**2. High Level Design**

Primary components that constitutes our game includes the ARM core (game logic), device driver, USB controller to control the input from keyboard, Sprite controller (control the display of sprites), audio controller, SDRAM ( store all the data needed in game logic). The game logic module interfaces with several other modules including the USB keyboard, by receiving the control signal; as well as the device driver in order to control the audio and display of sprites, including the positions of pillars and birds, the length of the pillars and the score. Sprite controller is connected to VGA Controller, which is responsible for the display of all the images, and audio controller is connected to audio CODEC on the Pygame board. Each of the components in our design will be discussed in detail below



**3. Game Logic Controller**

We implement the game logic by using C programming language. The game logic controller should realize the functions which are indicated below: updating the location of the bird from the keyboard, implementing the game rule (whether the game is over or not, computing how many pillars the bird has passed), generating the appropriate audio in terms of the game rule, and controlling the generation of sprites. Based on the functions given above, there should be 3 submodules for the game logic controller, the figure of which is shown below:



1. Game rules:

This is the core submodule of the game logic controller which interfaces with all of the other submodules, instructing them what to do based on the game rules. The rules are implemented by the updated position of bird from the keyboard, and the current position of the pillars. Appropriate audio is chosen corresponding to the rules (whether the game is over or not).

2. Sprite generator:

**a. Pillars:** This submodule keeps updating the X coordinates of the pillars that has already appeared on the screen (by decrementing them in every cycle), as well as the length of the upcoming pillar that is going to appear from the right side of the screen (which is actually the number of "partial" pillars that stack). The length of the pillar should be random, as long as the distance between the pillars is constant. Once the sprite moves out of the screen (in this case, x coordinate of any one of the pillars becomes zero), we reset the coordinate so that it can reappear from the right side of the screen.

**b. Bird:** Bird acts like in real world that its jump and fall will be affect by the gravity. When we implement the object motion formula in our code, time calculation is an issue that we use a counter counting instead of using system clock. We put the delay in our loop and try a suitable count number being our time unit. In addition, we add a status variable to indicate if the bird status is rising or falling. It cooperates with our jumping and falling function with iteration loop supporting continuous jumping without multi-thread.

**c. Score:** Every time the bird passes one of the pillars, the "Game Rules" submodule sends a signal, which will make the score increment by 1. Since the sprite for displaying the score are separated into 3 parts, hundreds, tens and digits, we need to extract them from the score before sending them to the hardware.

**d. Title:** The display of the title "Flappy Bird" depends on whether the game starts. When the game is over, press "enter" to restart, and the title would be displayed instantly. Since each signal sent from software to hardware has 8 bits, we only use one of them as the control signal to display the title, so that we can use other bits for other purposes, which improve efficiency

3.Audio generator: There are three audios to be played, one played

once pressing the "jump" button, and two played consecutively once the game is over. The selection of the audio is based on the signals from "Game Rule" submodule

**4.Game stuff preparation**

The preparations required for the graphics and audio are similar. First the image and audio files had to be searched for online. Once we agreed on the images and audio for the game, we edited them to fit our game design. Finally, both the image and audio files had to be converted to MIF format in order to be stored in the on-­‐chip ROM blocks.

Image preparation

For image preparation, first we resize the image to the size we will use. Then we do image segmentation of the images to separate the useful part. Then we set the background to a pure color so that the sprite controller can easily recognize the background part and remove it. Finally we convert the processed image to MIF files. In FPGA we will use different module to read the data of the MIF files. The image we processed is showed below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Blocks | background | bird | number | Pipe | sun |
| Numbers | 1 | 1 | 10 | 2 | 1 |
| Pixels | 128\*64 | 40\*40 | 51\*33 | 20\*125 | 50\*50 |
| ROM  size(bytes) | 24567 | 4800 | 1683 | 2500 | 7500 |
| example |  |  |  |  |  |

Audio preparation

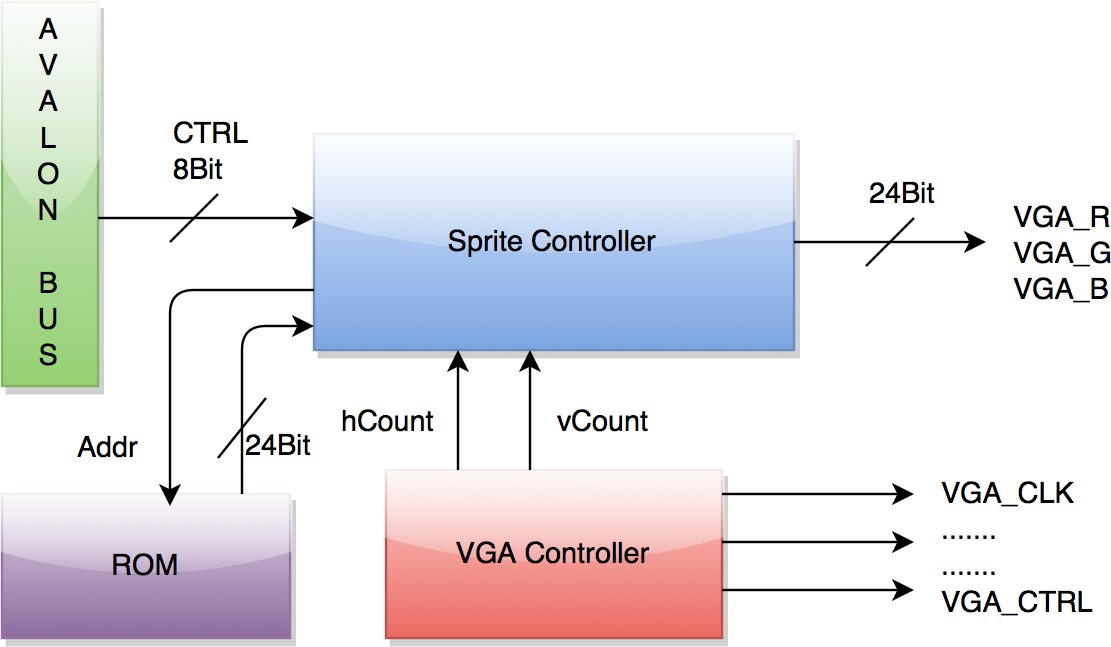
For audio preparation, it is familiar as the image processing. In this game, we totally used three sound segment: Super Mario jumping sound, Super Mario death sound and Doodle Jump falling down sound. For Super Mario death sound, it is a segment with 22050 Hz sampling rate and 65536 samples. The codec sampling rate of FPGA we use is 44100 Hz. So we first upsample the sound segment by a factor of 2. Then it becomes a segment of 131072 samples. The maximum of MegaWizard ROM words is 65536. Thus we should divide the segment into two parts each of them consists 65535 samples. Then convert them into MIF files. The process of jumping sound and falling down sound is same as the death sound.

**5.VGA Device Drive**

The VGA module is actually a memory-­‐mapped slave, which connects to the Avalon bus through the lightweight AXI bridge. The HPS uses 4-­‐bit address bits to access 16 location that store 8 bits data. More specifically, The software use ioctl to call the iowrite function in the device driver and specify the registers’ address(a base address of the vga slave plus the offset address which is specified in the device tree ) to write. We use Qsys to connect everything between vga\_led and the HPS up.

**6.Sprite Controllers and VGA Display**

VGA display is the core part of our project, VGA scan the screen and display pixels of graph. The video display controller has two major blocks, the VGA controller and the Sprite Controller. (see figure 4). Detailed introductions of Sprite Controller and VGA Controller are as following:



VGA Controller: This module generates the VGA signals needed by the VGA interface and also hcount and vcount values that are used in Sprite controller.

Sprite Controller: Based on the control signal received from the software, the sprite controller decides which sprites should be displayed and where they should be. Then sprite controller gets the data from the sprite ROMs and sends the RGB values of each pixel to the VGA interface. The inputs for the sprite controller are the following

Hcount and Vcount

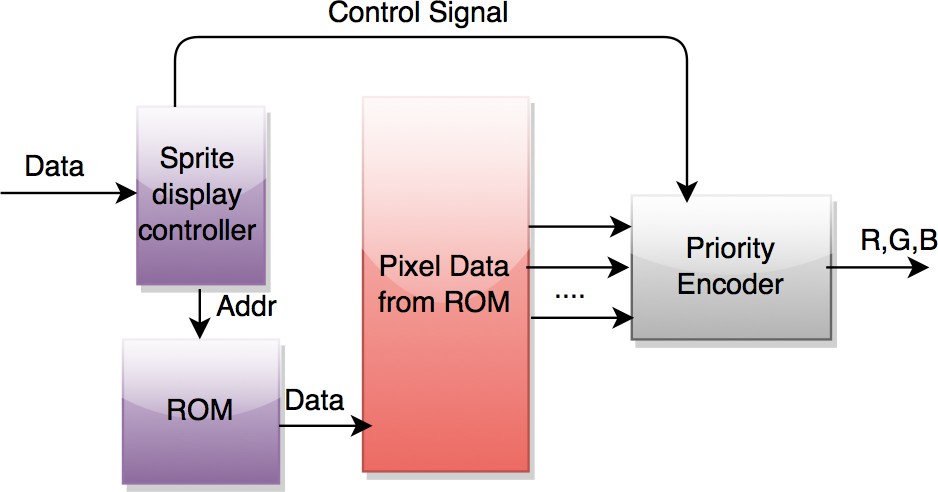
Position of bird

Position of pipe

Height of pipe

Game start

Score



We also implement priority encoder in the sprite controller. The game consists of 4 layers. The order of the layer is as follows:

The background layer has the lowest priority

The pipe layer comes next

The score layer is next

The topmost layer is the bird layer

Another problem about VGA is that the data should be updated at the vertical blanking time when the screen scanning reach to the area out of the screen. Other wise, if the data is changed during the scanning of the visible area of screen, the screen may be a little distorted. To avoid the distortion, we only update the value of data when the vertical scanning is beyond the v\_active region.

**7.Audio**

Flappy bird supports sound for bird jumping and gameover music.

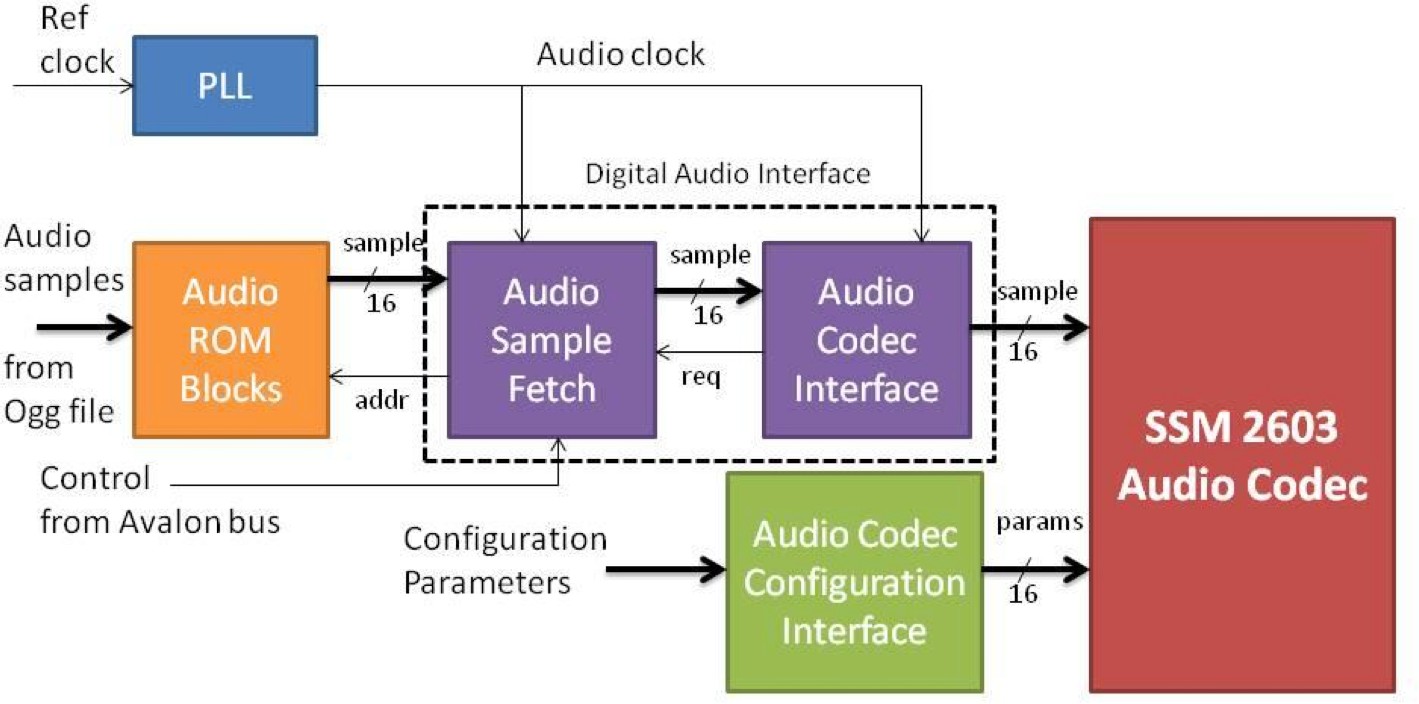
The audio controller has 3 main components:

1) Audio Data, 2) Audio codec configuration interface 3) Digital audio interface.

We use three sound files in this game. First we convert them to memory initialization file format. These .mif files for the jumping sound (jumping.mif), dead sound (dead.mif) and the falling down sound (death.mif) are used to create ROM data blocks using megawizard. Jumping and falling down music ROM blocks contain 32768 16-­‐bit audio samples and dead sound ROM block contains 65536 16-­‐bit audio samples. The total size of the memory used for audio storage is 128KB.

Audio Codec Configuration Interface is used to configure the various parameters inside the SSM2603 audio codec. This interface uses the I2C protocol to communicate the configuration parameters to the audio codec. Some of the configured parameters are: volume (which is set to 0 dB), the mode of the audio codec (which is set to slave), sampling rate (we are using 44.1 kHz), power on and off the audio codec, etc.

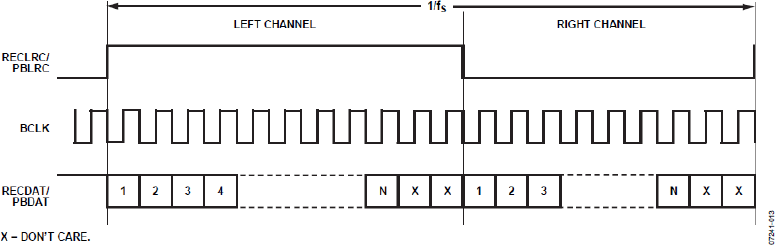
Digital Audio Interface has two sub-­‐components: a) Audio sample fetch and b) Audio codec interface. Both of these sub-­‐components operate at the audio clock rate (11.3 MHz), which is derived from the reference clock (50 MHz) using Phase Locked Loop (PLL). The complete block diagram is shown below.



The audio sample fetch is used to get the 16-­‐bit audio samples from the Audio ROM blocks, which are accessed using the address bits for the blocks. The fetch unit also takes control as input, which comes from the audio peripheral module in software. This control signal is used to control the switching on and off of the jumping and dead sound.

The Audio codec interface sub-­‐component sends audio samples to the audio codec using shift registers, that shift these samples at fixed clock rate. The audio clock is used to derive two audio clocks: (i) Left Right Channel (LRC) clock and (ii) Bit clock. Both these clocks are generated from the audio clock using clock divider.

The LRC clock is used for time multiplexing the audio samples. The audio sample can be sent out on the positive phase (left channel) of the clock or negative phase (right channel). The bit clock is used to send each bit of the audio sample as shown by the timing diagram in Figure 7. Please note as there are many number of cycles in one phase of the LRC clock, the codec interface sends don’t cares for the remaining cycles are after transmitting 16 bits of the audio sample.

**8.Source code**

**Flappy.py**

from itertools import cycle

import random

import sys

import pygame

from pygame.locals import \*

FPS = 30

SCREENWIDTH = 288

SCREENHEIGHT = 512

PIPEGAPSIZE = 100 # gap between upper and lower part of pipe

BASEY = SCREENHEIGHT \* 0.79

# image, sound and hitmask dicts

IMAGES, SOUNDS, HITMASKS = {}, {}, {}

# list of all possible players (tuple of 3 positions of flap)

PLAYERS\_LIST = (

# red bird

(

'assets/sprites/redbird-upflap.png',

'assets/sprites/redbird-midflap.png',

'assets/sprites/redbird-downflap.png',

),

# blue bird

(

'assets/sprites/bluebird-upflap.png',

'assets/sprites/bluebird-midflap.png',

'assets/sprites/bluebird-downflap.png',

),

# yellow bird

(

'assets/sprites/yellowbird-upflap.png',

'assets/sprites/yellowbird-midflap.png',

'assets/sprites/yellowbird-downflap.png',

),

)

# list of backgrounds

BACKGROUNDS\_LIST = (

'assets/sprites/background-day.png',

'assets/sprites/background-night.png',

)

# list of pipes

PIPES\_LIST = (

'assets/sprites/pipe-green.png',

'assets/sprites/pipe-red.png',

)

try:

xrange

except NameError:

xrange = range

def main():

global SCREEN, FPSCLOCK

pygame.init()

FPSCLOCK = pygame.time.Clock()

SCREEN = pygame.display.set\_mode((SCREENWIDTH, SCREENHEIGHT))

pygame.display.set\_caption('Flappy Bird')

# numbers sprites for score display

IMAGES['numbers'] = (

pygame.image.load('assets/sprites/0.png').convert\_alpha(),

pygame.image.load('assets/sprites/1.png').convert\_alpha(),

pygame.image.load('assets/sprites/2.png').convert\_alpha(),

pygame.image.load('assets/sprites/3.png').convert\_alpha(),

pygame.image.load('assets/sprites/4.png').convert\_alpha(),

pygame.image.load('assets/sprites/5.png').convert\_alpha(),

pygame.image.load('assets/sprites/6.png').convert\_alpha(),

pygame.image.load('assets/sprites/7.png').convert\_alpha(),

pygame.image.load('assets/sprites/8.png').convert\_alpha(),

pygame.image.load('assets/sprites/9.png').convert\_alpha()

)

# game over sprite

IMAGES['gameover'] = pygame.image.load('assets/sprites/gameover.png').convert\_alpha()

# message sprite for welcome screen

IMAGES['message'] = pygame.image.load('assets/sprites/message.png').convert\_alpha()

# base (ground) sprite

IMAGES['base'] = pygame.image.load('assets/sprites/base.png').convert\_alpha()

# sounds

if 'win' in sys.platform:

soundExt = '.wav'

else:

soundExt = '.ogg'

SOUNDS['die'] = pygame.mixer.Sound('assets/audio/die' + soundExt)

SOUNDS['hit'] = pygame.mixer.Sound('assets/audio/hit' + soundExt)

SOUNDS['point'] = pygame.mixer.Sound('assets/audio/point' + soundExt)

SOUNDS['swoosh'] = pygame.mixer.Sound('assets/audio/swoosh' + soundExt)

SOUNDS['wing'] = pygame.mixer.Sound('assets/audio/wing' + soundExt)

while True:

# select random background sprites

randBg = random.randint(0, len(BACKGROUNDS\_LIST) - 1)

IMAGES['background'] = pygame.image.load(BACKGROUNDS\_LIST[randBg]).convert()

# select random player sprites

randPlayer = random.randint(0, len(PLAYERS\_LIST) - 1)

IMAGES['player'] = (

pygame.image.load(PLAYERS\_LIST[randPlayer][0]).convert\_alpha(),

pygame.image.load(PLAYERS\_LIST[randPlayer][1]).convert\_alpha(),

pygame.image.load(PLAYERS\_LIST[randPlayer][2]).convert\_alpha(),

)

# select random pipe sprites

pipeindex = random.randint(0, len(PIPES\_LIST) - 1)

IMAGES['pipe'] = (

pygame.transform.flip(

pygame.image.load(PIPES\_LIST[pipeindex]).convert\_alpha(), False, True),

pygame.image.load(PIPES\_LIST[pipeindex]).convert\_alpha(),

)

# hitmask for pipes

HITMASKS['pipe'] = (

getHitmask(IMAGES['pipe'][0]),

getHitmask(IMAGES['pipe'][1]),

)

# hitmask for player

HITMASKS['player'] = (

getHitmask(IMAGES['player'][0]),

getHitmask(IMAGES['player'][1]),

getHitmask(IMAGES['player'][2]),

)

movementInfo = showWelcomeAnimation()

crashInfo = mainGame(movementInfo)

showGameOverScreen(crashInfo)

def showWelcomeAnimation():

"""Shows welcome screen animation of flappy bird"""

# index of player to blit on screen

playerIndex = 0

playerIndexGen = cycle([0, 1, 2, 1])

# iterator used to change playerIndex after every 5th iteration

loopIter = 0

playerx = int(SCREENWIDTH \* 0.2)

playery = int((SCREENHEIGHT - IMAGES['player'][0].get\_height()) / 2)

messagex = int((SCREENWIDTH - IMAGES['message'].get\_width()) / 2)

messagey = int(SCREENHEIGHT \* 0.12)

basex = 0

# amount by which base can maximum shift to left

baseShift = IMAGES['base'].get\_width() - IMAGES['background'].get\_width()

# player shm for up-down motion on welcome screen

playerShmVals = {'val': 0, 'dir': 1}

while True:

for event in pygame.event.get():

if event.type == QUIT or (event.type == KEYDOWN and event.key == K\_ESCAPE):

pygame.quit()

sys.exit()

if event.type == KEYDOWN and (event.key == K\_SPACE or event.key == K\_UP):

# make first flap sound and return values for mainGame

SOUNDS['wing'].play()

return {

'playery': playery + playerShmVals['val'],

'basex': basex,

'playerIndexGen': playerIndexGen,

}

# adjust playery, playerIndex, basex

if (loopIter + 1) % 5 == 0:

playerIndex = next(playerIndexGen)

loopIter = (loopIter + 1) % 30

basex = -((-basex + 4) % baseShift)

playerShm(playerShmVals)

# draw sprites

SCREEN.blit(IMAGES['background'], (0,0))

SCREEN.blit(IMAGES['player'][playerIndex],

(playerx, playery + playerShmVals['val']))

SCREEN.blit(IMAGES['message'], (messagex, messagey))

SCREEN.blit(IMAGES['base'], (basex, BASEY))

pygame.display.update()

FPSCLOCK.tick(FPS)

def mainGame(movementInfo):

score = playerIndex = loopIter = 0

playerIndexGen = movementInfo['playerIndexGen']

playerx, playery = int(SCREENWIDTH \* 0.2), movementInfo['playery']

basex = movementInfo['basex']

baseShift = IMAGES['base'].get\_width() - IMAGES['background'].get\_width()

# get 2 new pipes to add to upperPipes lowerPipes list

newPipe1 = getRandomPipe()

newPipe2 = getRandomPipe()

# list of upper pipes

upperPipes = [

{'x': SCREENWIDTH + 200, 'y': newPipe1[0]['y']},

{'x': SCREENWIDTH + 200 + (SCREENWIDTH / 2), 'y': newPipe2[0]['y']},

]

# list of lowerpipe

lowerPipes = [

{'x': SCREENWIDTH + 200, 'y': newPipe1[1]['y']},

{'x': SCREENWIDTH + 200 + (SCREENWIDTH / 2), 'y': newPipe2[1]['y']},

]

dt = FPSCLOCK.tick(FPS)/1000

pipeVelX = -128 \* dt

# player velocity, max velocity, downward acceleration, acceleration on flap

playerVelY = -9 # player's velocity along Y, default same as playerFlapped

playerMaxVelY = 10 # max vel along Y, max descend speed

playerMinVelY = -8 # min vel along Y, max ascend speed

playerAccY = 1 # players downward acceleration

playerRot = 45 # player's rotation

playerVelRot = 3 # angular speed

playerRotThr = 20 # rotation threshold

playerFlapAcc = -9 # players speed on flapping

playerFlapped = False # True when player flaps

while True:

for event in pygame.event.get():

if event.type == QUIT or (event.type == KEYDOWN and event.key == K\_ESCAPE):

pygame.quit()

sys.exit()

if event.type == KEYDOWN and (event.key == K\_SPACE or event.key == K\_UP):

if playery > -2 \* IMAGES['player'][0].get\_height():

playerVelY = playerFlapAcc

playerFlapped = True

SOUNDS['wing'].play()

# check for crash here

crashTest = checkCrash({'x': playerx, 'y': playery, 'index': playerIndex},

upperPipes, lowerPipes)

if crashTest[0]:

return {

'y': playery,

'groundCrash': crashTest[1],

'basex': basex,

'upperPipes': upperPipes,

'lowerPipes': lowerPipes,

'score': score,

'playerVelY': playerVelY,

'playerRot': playerRot

}

# check for score

playerMidPos = playerx + IMAGES['player'][0].get\_width() / 2

for pipe in upperPipes:

pipeMidPos = pipe['x'] + IMAGES['pipe'][0].get\_width() / 2

if pipeMidPos <= playerMidPos < pipeMidPos + 4:

score += 1

SOUNDS['point'].play()

# playerIndex basex change

if (loopIter + 1) % 3 == 0:

playerIndex = next(playerIndexGen)

loopIter = (loopIter + 1) % 30

basex = -((-basex + 100) % baseShift)

# rotate the player

if playerRot > -90:

playerRot -= playerVelRot

# player's movement

if playerVelY < playerMaxVelY and not playerFlapped:

playerVelY += playerAccY

if playerFlapped:

playerFlapped = False

# more rotation to cover the threshold (calculated in visible rotation)

playerRot = 45

playerHeight = IMAGES['player'][playerIndex].get\_height()

playery += min(playerVelY, BASEY - playery - playerHeight)

# move pipes to left

for uPipe, lPipe in zip(upperPipes, lowerPipes):

uPipe['x'] += pipeVelX

lPipe['x'] += pipeVelX

# add new pipe when first pipe is about to touch left of screen

if 3 > len(upperPipes) > 0 and 0 < upperPipes[0]['x'] < 5:

newPipe = getRandomPipe()

upperPipes.append(newPipe[0])

lowerPipes.append(newPipe[1])

# remove first pipe if its out of the screen

if len(upperPipes) > 0 and upperPipes[0]['x'] < -IMAGES['pipe'][0].get\_width():

upperPipes.pop(0)

lowerPipes.pop(0)

# draw sprites

SCREEN.blit(IMAGES['background'], (0,0))

for uPipe, lPipe in zip(upperPipes, lowerPipes):

SCREEN.blit(IMAGES['pipe'][0], (uPipe['x'], uPipe['y']))

SCREEN.blit(IMAGES['pipe'][1], (lPipe['x'], lPipe['y']))

SCREEN.blit(IMAGES['base'], (basex, BASEY))

# print score so player overlaps the score

showScore(score)

# Player rotation has a threshold

visibleRot = playerRotThr

if playerRot <= playerRotThr:

visibleRot = playerRot

playerSurface = pygame.transform.rotate(IMAGES['player'][playerIndex], visibleRot)

SCREEN.blit(playerSurface, (playerx, playery))

pygame.display.update()

FPSCLOCK.tick(FPS)

def showGameOverScreen(crashInfo):

"""crashes the player down and shows gameover image"""

score = crashInfo['score']

playerx = SCREENWIDTH \* 0.2

playery = crashInfo['y']

playerHeight = IMAGES['player'][0].get\_height()

playerVelY = crashInfo['playerVelY']

playerAccY = 2

playerRot = crashInfo['playerRot']

playerVelRot = 7

basex = crashInfo['basex']

upperPipes, lowerPipes = crashInfo['upperPipes'], crashInfo['lowerPipes']

# play hit and die sounds

SOUNDS['hit'].play()

if not crashInfo['groundCrash']:

SOUNDS['die'].play()

while True:

for event in pygame.event.get():

if event.type == QUIT or (event.type == KEYDOWN and event.key == K\_ESCAPE):

pygame.quit()

sys.exit()

if event.type == KEYDOWN and (event.key == K\_SPACE or event.key == K\_UP):

if playery + playerHeight >= BASEY - 1:

return

# player y shift

if playery + playerHeight < BASEY - 1:

playery += min(playerVelY, BASEY - playery - playerHeight)

# player velocity change

if playerVelY < 15:

playerVelY += playerAccY

# rotate only when it's a pipe crash

if not crashInfo['groundCrash']:

if playerRot > -90:

playerRot -= playerVelRot

# draw sprites

SCREEN.blit(IMAGES['background'], (0,0))

for uPipe, lPipe in zip(upperPipes, lowerPipes):

SCREEN.blit(IMAGES['pipe'][0], (uPipe['x'], uPipe['y']))

SCREEN.blit(IMAGES['pipe'][1], (lPipe['x'], lPipe['y']))

SCREEN.blit(IMAGES['base'], (basex, BASEY))

showScore(score)

playerSurface = pygame.transform.rotate(IMAGES['player'][1], playerRot)

SCREEN.blit(playerSurface, (playerx,playery))

SCREEN.blit(IMAGES['gameover'], (50, 180))

FPSCLOCK.tick(FPS)

pygame.display.update()

def playerShm(playerShm):

"""oscillates the value of playerShm['val'] between 8 and -8"""

if abs(playerShm['val']) == 8:

playerShm['dir'] \*= -1

if playerShm['dir'] == 1:

playerShm['val'] += 1

else:

playerShm['val'] -= 1

def getRandomPipe():

"""returns a randomly generated pipe"""

# y of gap between upper and lower pipe

gapY = random.randrange(0, int(BASEY \* 0.6 - PIPEGAPSIZE))

gapY += int(BASEY \* 0.2)

pipeHeight = IMAGES['pipe'][0].get\_height()

pipeX = SCREENWIDTH + 10

return [

{'x': pipeX, 'y': gapY - pipeHeight}, # upper pipe

{'x': pipeX, 'y': gapY + PIPEGAPSIZE}, # lower pipe

]

def showScore(score):

"""displays score in center of screen"""

scoreDigits = [int(x) for x in list(str(score))]

totalWidth = 0 # total width of all numbers to be printed

for digit in scoreDigits:

totalWidth += IMAGES['numbers'][digit].get\_width()

Xoffset = (SCREENWIDTH - totalWidth) / 2

for digit in scoreDigits:

SCREEN.blit(IMAGES['numbers'][digit], (Xoffset, SCREENHEIGHT \* 0.1))

Xoffset += IMAGES['numbers'][digit].get\_width()

def checkCrash(player, upperPipes, lowerPipes):

"""returns True if player collides with base or pipes."""

pi = player['index']

player['w'] = IMAGES['player'][0].get\_width()

player['h'] = IMAGES['player'][0].get\_height()

# if player crashes into ground

if player['y'] + player['h'] >= BASEY - 1:

return [True, True]

else:

playerRect = pygame.Rect(player['x'], player['y'],

player['w'], player['h'])

pipeW = IMAGES['pipe'][0].get\_width()

pipeH = IMAGES['pipe'][0].get\_height()

for uPipe, lPipe in zip(upperPipes, lowerPipes):

# upper and lower pipe rects

uPipeRect = pygame.Rect(uPipe['x'], uPipe['y'], pipeW, pipeH)

lPipeRect = pygame.Rect(lPipe['x'], lPipe['y'], pipeW, pipeH)

# player and upper/lower pipe hitmasks

pHitMask = HITMASKS['player'][pi]

uHitmask = HITMASKS['pipe'][0]

lHitmask = HITMASKS['pipe'][1]

# if bird collided with upipe or lpipe

uCollide = pixelCollision(playerRect, uPipeRect, pHitMask, uHitmask)

lCollide = pixelCollision(playerRect, lPipeRect, pHitMask, lHitmask)

if uCollide or lCollide:

return [True, False]

return [False, False]

def pixelCollision(rect1, rect2, hitmask1, hitmask2):

"""Checks if two objects collide and not just their rects"""

rect = rect1.clip(rect2)

if rect.width == 0 or rect.height == 0:

return False

x1, y1 = rect.x - rect1.x, rect.y - rect1.y

x2, y2 = rect.x - rect2.x, rect.y - rect2.y

for x in xrange(rect.width):

for y in xrange(rect.height):

if hitmask1[x1+x][y1+y] and hitmask2[x2+x][y2+y]:

return True

return False

def getHitmask(image):

"""returns a hitmask using an image's alpha."""

mask = []

for x in xrange(image.get\_width()):

mask.append([])

for y in xrange(image.get\_height()):

mask[x].append(bool(image.get\_at((x,y))[3]))

return mask

if \_\_name\_\_ == '\_\_main\_\_':

main()

**setup.py**

import os

import sys

from distutils.core import setup

import py2exe

origIsSystemDLL = py2exe.build\_exe.isSystemDLL

def isSystemDLL(pathname):

dlls = ("libfreetype-6.dll", "libogg-0.dll", "sdl\_ttf.dll")

if os.path.basename(pathname).lower() in dlls:

return 0

return origIsSystemDLL(pathname)

py2exe.build\_exe.isSystemDLL = isSystemDLL

sys.argv.append('py2exe')

setup(

name = 'Flappy Bird',

version = '1.0',

author = 'Garv Kumar',

options = {

'py2exe': {

'bundle\_files': 1, # doesn't work on win64

'compressed': True,

}

},

windows = [{

'script': "flappy.py",

'icon\_resources': [

(1, 'flappy.ico')

]

}],

zipfile=None,

)