**(SPACE INVADERS)**

Mini Project Report of

Open Source Technology Lab



Shreyash Karandikar (Roll No. 17102C2006)

Pranav Badgi (Roll No. 17102C2007)

Pratik Bhavsar (Roll No. 17102c2010)

Department of Computer Engineering

Vidyalankar Institute of Technology

Wadala(E), Mumbai- 400437

University of Mumbai

2017-18

Mini Project Report Approval for “Open Source Technology Lab”.

This mini project report entitled **SPACE INVADERS** by

1. Shreyash Karandikar (17102C2006)

2. Pranav Badgi (17102C2007)

3. Pratik Bhavsar (17102C2010)

have successfully completed for the course “Open Source Technology Lab”.

Signature of Subject Incharge

---------------------------------------------

Date:

Place:

**ABSTRACT**

Space Invaders is a two-dimensional fixed shooter game in which the player controls a ship with lasers by moving it horizontally across the bottom of the screen and firing at descending aliens. The aim is to defeat five rows of ten aliens that move horizontally back and forth across the screen as they advance towards the bottom of the screen. The player defeats an alien, and earns points, by shooting it with the laser cannon. As more aliens are defeated, the aliens’ movement and the game’s music both speed up.

**INTRODUCTION**

Space Adventures is a game based on the original Space Invaders arcade game created by Tomohiro Nishikado in 1978. The objective of the game is to shoot enemy drones as they descend downwards towards the hero. The hero spaceship is equipped with a canon that fires towards enemy ships as well as a bonus ship that hovers above the enemy ships. The score updates itself, as more enemy ships are killed. The hero has three lives and after losing all the lives the game ends. However, if the hero is able to shoot all the enemies the game moves on to the next round.

**EXPLANATION**

**MODULES USED**

**turtle:** Turtle graphics is a popular way for introducing programming to kids. It was part of the original Logo programming language developed by Wally Feurzig and Seymour Papert in 1966.

The [**turtle**](https://docs.python.org/2/library/turtle.html#module-turtle) module is an extended reimplementation of the same-named module from the Python standard distribution up to version Python 2.5.

It tries to keep the merits of the old turtle module and to be (nearly) 100% compatible with it. This means in the first place to enable the learning programmer to use all the commands, classes and methods interactively when using the module from within IDLE run with the -n switch.

The turtle module provides turtle graphics primitives, in both object-oriented and procedure-oriented ways. Because it uses **[Tkinter](https://docs.python.org/2/library/tkinter.html" \l "module-Tkinter" \o "Tkinter: Interface to Tcl/Tk for graphical user interfaces)** for the underlying graphics, it needs a version of Python installed with Tk support.

**OS Module:** The OS module in python provides a way of using operating system dependent functionality.

The function that the OS module provides allows you to interface with the underlying operating system that python is running on – be that Windows, Mac or Linux.

You can find important information about the location or about the processes.

**Random Library:** This module implements pseudo-random number generators for various distributions.

For integers, uniform selection from a range. For sequences, uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.

On the real line, there are functions to compute uniform, normal (Gaussian), lognormal, negative exponential, gamma, and beta distributions. For generating distributions of angles, the von Mises distribution is available.

**Sys:** System-specific parameters and functions. This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available. ... If no script name was passed to the **Python** interpreter, argv[0] is the empty string.

**CODE:**

import turtle

import os

import math

import random

import sys

#Set up the screen

wn = turtle.Screen()

wn.bgcolor("black")

wn.title("Space Invaders")

wn.bgpic("space\_invaders\_background.gif")

#register the shapes

turtle.register\_shape("invader.gif")

turtle.register\_shape("player.gif")

#Draw border

border\_pen = turtle.Turtle()

border\_pen.speed(0)

border\_pen.color("white")

border\_pen.penup()

border\_pen.setposition(-300,-300)

border\_pen.pendown()

border\_pen.pensize(3)

for side in range(4):

border\_pen.fd(600)

border\_pen.lt(90)

border\_pen.hideturtle()

#set the score to 0

score = 0

#draw the score

score\_pen = turtle.Turtle()

score\_pen.speed(0)

score\_pen.color("white")

score\_pen.penup()

score\_pen.setposition(-290,280)

scorestring = "Score: %s" %score

score\_pen.write(scorestring,False,align ="left",font=("Arial",14,"normal"))

score\_pen.hideturtle()

#Create the player turtle

player = turtle.Turtle()

player.color("blue")

player.shape("player.gif")

player.penup()

player.speed(0)

player.setposition(0, -250)

player.setheading(90)

playerspeed = 15

#Choose a number of enemies

number\_of\_enemies = 8

#Create an empty list of enemies

enemies = []

#Add enemies to the list

for i in range(number\_of\_enemies):

#Create the enemy

enemies.append(turtle.Turtle())

for enemy in enemies:

enemy.color("red")

enemy.shape("invader.gif")

enemy.penup()

enemy.speed(0)

x = random.randint(-200, 200)

y = random.randint(100, 250)

enemy.setposition(x, y)

enemyspeed = 5

#Create the player's bullet

bullet = turtle.Turtle()

bullet.color("yellow")

bullet.shape("triangle")

bullet.penup()

bullet.speed(0)

bullet.setheading(90)

bullet.shapesize(0.5, 0.5)

bullet.hideturtle()

bulletspeed = 30

#Define bullet state

#ready - ready to fire

#fire - bullet is firing

bulletstate = "ready"

#Move the player left and right

def move\_left():

x = player.xcor()

x -= playerspeed

if x < -280:

x = - 280

player.setx(x)

def move\_right():

x = player.xcor()

x += playerspeed

if x > 280:

x = 280

player.setx(x)

def fire\_bullet():

#Declare bulletstate as a global if it needs changed

global bulletstate

if bulletstate == "ready":

bulletstate = "fire"

#Move the bullet to the just above the player

x = player.xcor()

y = player.ycor() + 10

bullet.setposition(x, y)

bullet.showturtle()

def isCollision(t1, t2):

distance = math.sqrt(math.pow(t1.xcor()-t2.xcor(),2)+math.pow(t1.ycor()-t2.ycor(),2))

if distance < 15:

return True

else:

return False

#Create keyboard bindings

turtle.listen()

turtle.onkey(move\_left, "Left")

turtle.onkey(move\_right, "Right")

turtle.onkey(fire\_bullet, "space")

#Main game loop

while True:

for enemy in enemies:

#Move the enemy

x = enemy.xcor()

x += enemyspeed

enemy.setx(x)

#invadors out of range

if enemy.ycor() < -220:

sys.exit()

#Move the enemy back and down

if enemy.xcor() > 280:

#Move all enemies down

for e in enemies:

y = e.ycor()

y -= 40

e.sety(y)

#Change enemy direction

enemyspeed \*= -1

if enemy.xcor() < -280:

#Move all enemies down

for e in enemies:

y = e.ycor()

y -= 40

e.sety(y)

#Change enemy direction

enemyspeed \*= -1

#Check for a collision between the bullet and the enemy

if isCollision(bullet, enemy):

#Reset the bullet

bullet.hideturtle()

bulletstate = "ready"

bullet.setposition(0, -400)

#Reset the enemy

x = random.randint(-200, 200)

y = random.randint(100, 250)

enemy.setposition(x, y)

#update the score

score += 10

scorestring = "score: %s" %score

score\_pen.clear()

score\_pen.write(scorestring,False,align="left",font=("Arial",14,"normal"))

if isCollision(player, enemy):

player.hideturtle()

enemy.hideturtle()

sys.exit()

print ("Game Over")

break

#Move the bullet

if bulletstate == "fire":

y = bullet.ycor()

y += bulletspeed

bullet.sety(y)

#Check to see if the bullet has gone to the top

if bullet.ycor() > 275:

bullet.hideturtle()

bulletstate = "ready"

delay = raw\_input("Press enter to finsh.")

**CONCLUSION**

The project was exciting and we learned a lot about handheld device programming. The application itself taught us how making games isn’t as difficult as we had initially envisioned. We initially designed the look and feel of the application, and then architected some of the classes. We worked on each class separately, and then brought together. The final application worked well in the demo and we felt satisfied.