# A Project Report on

**Game: Space Invaders 2**

Submitted by

**Burhanuddin Naguthanawala (19IT5001)**

**Ashish Patil (19IT5008)**

**Rithvik Chavhan (19IT5012)**

Under the guidance of

**Mrs. Anita Senathi Assistant Professor**

**Department of Information Technology Ramrao Adik Institute of Technology**

**Nerul, Navi Mumbai**

**Certificate**

This is to certify that the project entitled **Game: Space Invaders 2** is being submitted to the Department of Information Technology, Ramrao Adik Institute of Technology,Navi Mumbai.

Project Guide External

Examiner (Mrs. Anita Senathi) ( )

# Acknowledgement

We owe our gratitude to many people who have supported us throughout this journey.

We would, firstly like to express our heartfelt gratitude towards our respected Principal Dr. M.D. Patil and our Head of Department Dr. Ashish Jadhav for providing us immense facilities, guidance and never-ending support.

The completion of any inter-disciplinary project depends upon cooperation and combined efforts of several sources of knowledge. We take this opportunity to express our profound gratitude and deep regards to our guide Mr. Madhav Vyas for his exemplary guidance, monitoring and constant encouragement throughout the course of this project.

Lastly, we thank our parents, family, friends and well-wishers who always looked for the chance to help us in whatever means came forth and for their constant encouragement without which the project would not be a distant reality.

# Introduction

Space Invaders 2 is a user interactive offline game made using Python language.

In this game, the user plays as an spaceship that shoots space monsters that appear on the game screen.

There are total of 6 monsters that appear on the screen, to make it a bit interesting, when the monster collide with each other they spawn at random location.

When the monsters manage to reach the same position as the spaceship the game ends.

# Proposed System

The objective is to create a simple yet interesting game which has optimal accessibility and cost in terms of memory and execution. The game will be based on cognitive response.

The game will have 3 main parts, namely, the random generator and the math.

This game takes minimum space and time to run on an operating system while providing the best experience.

**Language:** Python

**Modules:** Pygame, Random, Math

* 1. **Pygame:**

**Pygame** is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language. Pygame was officially written by **Pete Shinners** to replace PySDL. Pygame is suitable to create client-side applications that can be potentially wrapped in a standalone executable. Applications using Pygame can run on Android phones and tablets with the use of Pygame Subset for Android (pgs4a). Sound, vibration, keyboard, and accelerometer are supported on Android.

* 1. **Random:**

Python offers random module that can generate random numbers.

These are pseudo-random number as the sequence of number generated depends on the seed. If the seeding value is same, the sequence will be the same. For example, if you use 2 as the seeding value, you will always see the following sequence. It is used **cryptographically as secure** random generator using a **secrets module to** generate secure **tokens**, security **keys**, and **URLs.**

* 1. **Math:**

This module provides access to the mathematical functions defined by the C standard.These functions cannot be used with complex numbers; use the functions of the same name from the [cmath](https://docs.python.org/3/library/cmath.html" \l "module-cmath" \o "cmath: Mathematical functions for complex numbers.) module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don’t is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place

**WORKING OF SYSTEM**

Space Invaders 2 works on: -

1. **Intro:**

The main game screen is of 800x600. The game consists of 2 characters, the spaceship (user) and monsters. The game starts as soon as the user moves.

The game ends as soon as the monsters reach the user.

1. **Collision Logic:**

The collision helps to spawn the monsters on random spots as soon as they collide with each other on the widow, this makes the game challenging. Total of 6 monsters can spawn at random spots if they collide with each other.

# Advantages of Space Invaders 2:

1. Easy to implement.
2. User friendly.
3. Less disk space required.
4. Optimal accessibility and cost in terms of memory and speed.

Source code:

|  |
| --- |
|  |
|  | import math  import random |
|  |  |
|  | import pygame |
|  | from pygame import mixer |
|  |  |
|  | # Intialize the pygame |
|  | pygame.init() |
|  |  |
|  | # create the screen |
|  | screen = pygame.display.set\_mode((800, 600)) |
|  |  |
|  | # Background |
|  | background = pygame.image.load('back3.png') |
|  |  |
|  |  |
|  | # Caption and Icon |
|  | pygame.display.set\_caption("Space Invader") |
|  | icon = pygame.image.load('rocket.png') |
|  | pygame.display.set\_icon(icon) |
|  |  |
|  | # Player |
|  | playerImg = pygame.image.load('spaceship3.png') |
|  | playerX = 370 |
|  | playerY = 480 |
|  | playerX\_change = 0 |
|  |  |
|  | # Enemy |
|  | enemyImg = [] |
|  | enemyX = [] |
|  | enemyY = [] |
|  | enemyX\_change = [] |
|  | enemyY\_change = [] |
|  | num\_of\_enemies = 20 |
|  |  |
|  | for i in range(num\_of\_enemies): |
|  | enemyImg.append(pygame.image.load('enemy.png')) |
|  | enemyX.append(random.randint(0, 736)) |
|  | enemyY.append(random.randint(50, 150)) |
|  | enemyX\_change.append(4) |
|  | enemyY\_change.append(40) |
|  |  |
|  | # Bullet |
|  |  |
|  | # Ready - You can't see the bullet on the screen |
|  | # Fire - The bullet is currently moving |
|  |  |
|  | bulletImg = pygame.image.load('bullet.png') |
|  | bulletX = 0 |
|  | bulletY = 480 |
|  | bulletX\_change = 0 |
|  | bulletY\_change = 10 |
|  | bullet\_state = "ready" |
|  |  |
|  | # Score |
|  |  |
|  | score\_value = 0 |
|  | font = pygame.font.Font('freesansbold.ttf', 32) |
|  |  |
|  | textX = 10 |
|  | testY = 10 |
|  |  |
|  | # Game Over |
|  | over\_font = pygame.font.Font('freesansbold.ttf', 64) |
|  |  |
|  |  |
|  | def show\_score(x, y): |
|  | score = font.render("Score : " + str(score\_value), True, (255, 255, 255)) |
|  | screen.blit(score, (x, y)) |
|  |  |
|  |  |
|  | def game\_over\_text(): |
|  | over\_text = over\_font.render("GAME OVER", True, (255, 255, 255)) |
|  | screen.blit(over\_text, (200, 250)) |
|  |  |
|  |  |
|  | def player(x, y): |
|  | screen.blit(playerImg, (x, y)) |
|  |  |
|  |  |
|  | def enemy(x, y, i): |
|  | screen.blit(enemyImg[i], (x, y)) |
|  |  |
|  |  |
|  | def fire\_bullet(x, y): |
|  | global bullet\_state |
|  | bullet\_state = "fire" |
|  | screen.blit(bulletImg, (x + 16, y + 10)) |
|  |  |
|  |  |
|  | def isCollision(enemyX, enemyY, bulletX, bulletY): |
|  | distance = math.sqrt(math.pow(enemyX - bulletX, 2) + (math.pow(enemyY - bulletY, 2))) |
|  | if distance < 27: |
|  | return True |
|  | else: |
|  | return False |
|  |  |
|  |  |
|  | # Game Loop |
|  | running = True |
|  | while running: |
|  |  |
|  | # RGB = Red, Green, Blue |
|  | screen.fill((0, 0, 0)) |
|  | # Background Image |
|  | screen.blit(background, (0, 0)) |
|  | for event in pygame.event.get(): |
|  | if event.type == pygame.QUIT: |
|  | running = False |
|  | #bgm = mixer.Sound("bgm.wav") |
|  | #bgm.play() |
|  | # if keystroke is pressed check whether its right or left |
|  | if event.type == pygame.KEYDOWN: |
|  | if event.key == pygame.K\_LEFT: |
|  | playerX\_change = -9 |
|  | if event.key == pygame.K\_RIGHT: |
|  | playerX\_change = 9 |
|  | if event.key == pygame.K\_SPACE: |
|  | if bullet\_state is "ready": |
|  | bulletSound = mixer.Sound("laser.wav") |
|  | bulletSound.play() |
|  | # Get the current x cordinate of the spaceship |
|  | bulletX = playerX |
|  | fire\_bullet(bulletX, bulletY) |
|  |  |
|  | if event.type == pygame.KEYUP: |
|  | if event.key == pygame.K\_LEFT or event.key == pygame.K\_RIGHT: |
|  | playerX\_change = 0 |
|  |  |
|  | # 5 = 5 + -0.1 -> 5 = 5 - 0.1 |
|  | # 5 = 5 + 0.1 |
|  |  |
|  | playerX += playerX\_change |
|  | if playerX <= 0: |
|  | playerX = 0 |
|  | elif playerX >= 736: |
|  | playerX = 736 |
|  |  |
|  | # Enemy Movement |
|  | for i in range(num\_of\_enemies): |
|  |  |
|  | # Game Over |
|  | if enemyY[i] > 440: |
|  | for j in range(num\_of\_enemies): |
|  | enemyY[j] = 2000 |
|  | game\_over\_text() |
|  | break |
|  |  |
|  | enemyX[i] += enemyX\_change[i] |
|  | if enemyX[i] <= 0: |
|  | enemyX\_change[i] = 4 |
|  | enemyY[i] += enemyY\_change[i] |
|  | elif enemyX[i] >= 736: |
|  | enemyX\_change[i] = -4 |
|  | enemyY[i] += enemyY\_change[i] |
|  |  |
|  | # Collision |
|  | collision = isCollision(enemyX[i], enemyY[i], bulletX, bulletY) |
|  | if collision: |
|  | explosionSound = mixer.Sound("explosion.wav") |
|  | explosionSound.play() |
|  | bulletY = 480 |
|  | bullet\_state = "ready" |
|  | score\_value += 1 |
|  | enemyX[i] = random.randint(0, 736) |
|  | enemyY[i] = random.randint(50, 150) |
|  |  |
|  | enemy(enemyX[i], enemyY[i], i) |
|  |  |
|  | # Bullet Movement |
|  | if bulletY <= 0: |
|  | bulletY = 480 |
|  | bullet\_state = "ready" |
|  |  |
|  | if bullet\_state is "fire": |
|  | fire\_bullet(bulletX, bulletY) |
|  | bulletY -= bulletY\_change |
|  |  |
|  | player(playerX, playerY) |
|  | show\_score(textX, testY) |
|  | pygame.display.update() |

**Screenshots of the game:**

