Table of Contents

[Question 1 2](#_Toc180062278)

[Question 2 6](#_Toc180062279)

[Question 3 11](#_Toc180062280)

[Question 4 16](#_Toc180062281)

[4.1. 16](#_Toc180062282)

[4.2. 18](#_Toc180062283)

[References 20](#_Toc180062284)

## Question 1

1. Game Concept Overview

"Space Sniper" is a fast-paced, retro-style 2D arcade shooter where players control a spaceship, navigating a starfield while avoiding enemies and destroying them with missiles. The player must avoid collisions with the enemies, protect the allies, and survive as long as possible to achieve the highest score. The controls are limited to keyboard-only inputs.

2. Tools Required (Moraneus, 2024)

* Development Tools - The primary language will be Python. Pygame & PyOpenGL will be used for 2D game mechanics and rendering. The Turtle module will be used for graphics and animations throughout the game
* Sound Design Tools - Audacity for background sound
* Graphics Tools - GIMP or Aseprite will be used for sprite and background design

3. Target Devices

* Platform - Desktop can include Windows, macOS, or Linux.
* Reason - Pygame and Turtle modules (Python, n.d.) work efficiently on desktop systems, and a retro game experience is more immersive on larger screens.
* Input - Keyboard-only interaction ensures focus and simplicity.

4. Target Audience

* Primary Audience – Users that range from 12–30 years old. Users that are fans of retro games, arcade-style shooters, and quick action games.
* Secondary Audience - Casual gamers who enjoy short, engaging gameplay sessions. The game’s simple controls and fast-paced mechanics will appeal to players looking for nostalgia and easy-to-learn, as well as difficult-to-master games to play.

5. Stylistic Choices

* Visual Style:
  + Minimalist 2D graphics with pixel art sprites and bright colors against a dark space background.
  + Retro design evoking the look and feel of 80s arcade games (e.g., neon colors and simple shapes).
* Sound:
  + Simple sound effects (explosions, missile launches) using 8-bit-style audio clips for retro vibes.
  + Background music - A looping space soundtrack to induce the feeling of old-school games.

6. Interaction and Controls

* Keyboard Controls:
  + Arrow Keys - Control movement (the left and right arrows rotate the ship, the up arrow accelerates, and the down arrow decelerates)
  + Spacebar - Fire missile
* Keeping controls simple ensures accessibility and aligns with the retro experience.

7. Gameplay Mechanics

* Objective - Survive as long as possible by shooting enemies and protecting allies to earn the highest score.
* Enemies - Move randomly across the screen. Each collision with an enemy results in a score penalty and possible loss of a life.
* Allies - Friendly units that the player should avoid shooting. Hitting an ally decreases the player’s score.
* Missile Mechanics - Players can only fire one missile at a time. Once it moves off-screen or hits a target, it resets and becomes ready for the next shot.

8. End-Game Goal and Game Progression

* Goal - Maximize the player’s score before they run out of lives.
* Game Progression - The game gets harder with time: enemies move faster, and more allies appear, making it more challenging to maintain a high score.
  + Lives - The player starts with 3 lives. Each collision with an enemy costs a life, and the game ends when all lives are lost.

9. Scene Management

* Main Scenes:
  1. Main Menu - Start Game, Instructions, Quit
  2. Gameplay Scene - Starfield background with the player's spaceship, enemies, and allies.
  3. Game Over Screen - Displays final score and option to restart or quit.

10. Rendering Pipeline

* 2D Rendering Pipeline:
  + Turtle module will render objects such as the spaceship, enemies, and allies frame by frame.
  + The background (starfield) will remain static while sprites move smoothly using manual screen updates (using turtle.update()).
* Frame Rate:
  + The game will target 50 FPS, achieved by controlling update frequency with time.sleep(0.02).

11. Monetization Strategy

* Free-to-play model with the potential for future monetization via (Berrena, 2023):
  + In-game Ads - Non-intrusive banner ads on the main menu or between gameplay sessions.
  + Paid Versions - An ad-free version or optional cosmetic skins (e.g., spaceship colours) for small fees.

12. Justification of Design Decisions

* Retro Style - The pixel-art graphics, simple mechanics, and keyboard-only input align with the retro gaming theme.
* Accessibility - Limited controls make the game easy to learn and play, even for casual gamers.
* Replayability - The increasing difficulty ensures players are motivated to keep improving their high scores.
* Monetization - Keeping the game free-to-play initially will help build a user base, with optional paid features for dedicated players.

13. Mood Board

To capture the visual and thematic inspiration for "Space Sniper," the mood board includes:

* Colours - Neon blues, reds, and yellows on a black background.
* Visual Inspiration - Pixel art spaceships, starfields, and 80s-style UI elements.
* Sound Inspiration - synthwave music.

## Question 2

(Geeks for Geeks, 2020), (Geeks for Geeks, 2024), (Pygame, n.d.)

import os

import random

import time

import pygame

# Import the turtle graphics module for game visuals

import turtle

# Initialize pygame for sound management

pygame.init()

pygame.mixer.init()  # Initialize the mixer for sound

# Play background music in a loop

pygame.mixer.music.load("space.mp3")  # Load the music file

pygame.mixer.music.play(-1)  # -1 ensures the music loops indefinitely

# Configure the turtle window

turtle.speed(0)              # Max animation speed

turtle.bgcolor("black")      # Set background color to black

turtle.title("Space Sniper") # Title of the game window

turtle.bgpic("starfield.gif") # Load and display background image

turtle.ht()                  # Hide the default turtle cursor

turtle.setundobuffer(1)      # Set undo buffer size to 1 (optimize memory usage)

turtle.tracer(0)             # Disable automatic screen updates (manual updates for smoother animation)

# Define the base class for all game sprites

class Sprite(turtle.Turtle):

    def \_\_init\_\_(self, shape\_type, color, initial\_x, initial\_y):

        super().\_\_init\_\_(shape=shape\_type)  # Initialize as a turtle object

        self.speed(0)         # Max speed (instant movement)

        self.penup()           # Disable drawing when moving

        self.color(color)      # Set the color of the sprite

        self.goto(initial\_x, initial\_y)  # Place sprite at the starting position

        self.speed = 1         # Default movement speed

    def move(self):

        self.fd(self.speed)  # Move sprite forward based on speed

        # Boundary detection to keep sprite within the game area

        if self.xcor() > 290:

            self.setx(290)

            self.rt(60)  # Turn right upon hitting a boundary

        if self.xcor() < -290:

            self.setx(-290)

            self.rt(60)

        if self.ycor() > 290:

            self.sety(290)

            self.rt(60)

        if self.ycor() < -290:

            self.sety(-290)

            self.rt(60)

    def is\_collision(self, other):

        # Check for collision with another sprite (within 20 units distance)

        if (self.xcor() >= other.xcor() - 20) and \

           (self.xcor() <= other.xcor() + 20) and \

           (self.ycor() >= other.ycor() - 20) and \

           (self.ycor() <= other.ycor() + 20):

            return True

        return False

# Define the Player class (inherits from Sprite)

class Player(Sprite):

    def \_\_init\_\_(self, shape\_type, color, initial\_x, initial\_y):

        super().\_\_init\_\_(shape\_type, color, initial\_x, initial\_y)

        self.shapesize(stretch\_wid=0.6, stretch\_len=1.1)  # Adjust shape size

        self.speed = 4  # Set initial speed

        self.lives = 3  # Player starts with 3 lives

    # Define player controls

    def turn\_left(self):

        self.lt(45)  # Rotate left by 45 degrees

    def turn\_right(self):

        self.rt(45)  # Rotate right by 45 degrees

    def accelerate(self):

        self.speed += 1  # Increase speed

    def decelerate(self):

        self.speed -= 1  # Decrease speed

# Define Enemy class (inherits from Sprite)

class Enemy(Sprite):

    def \_\_init\_\_(self, shape\_type, color, initial\_x, initial\_y):

        super().\_\_init\_\_(shape\_type, color, initial\_x, initial\_y)

        self.speed = 6  # Set higher speed for enemy\_fleet

        self.setheading(random.randint(0, 360))  # Random movement direction

# Define Missile class (inherits from Sprite)

class Missile(Sprite):

    def \_\_init\_\_(self, shape\_type, color, initial\_x, initial\_y):

        super().\_\_init\_\_(shape\_type, color, initial\_x, initial\_y)

        self.shapesize(stretch\_wid=0.2, stretch\_len=0.4)  # Adjust missile size

        self.speed = 20  # Fast speed for missiles

        self.status = "ready"  # Track if missile is ready to fire

        self.goto(-1000, 1000)  # Hide missile off-screen initially

    def fire(self):

        if self.status == "ready":

            # Fire the missile from the player's position

            self.goto(player.xcor(), player.ycor())

            self.setheading(player.heading())  # Follow player's direction

            self.status = "firing"

    def move(self):

        if self.status == "firing":

            self.fd(self.speed)  # Move forward if fired

            # Reset if it goes out of bounds

            if self.xcor() < -290 or self.xcor() > 290 or \

               self.ycor() < -290 or self.ycor() > 290:

                self.goto(-1000, 1000)

                self.status = "ready"

# Set up keyboard controls

turtle.onkey(player.turn\_left, "Left")

turtle.onkey(player.turn\_right, "Right")

turtle.onkey(player.accelerate, "Up")

turtle.onkey(player.decelerate, "Down")

turtle.onkey(missile.fire, "space")

turtle.listen()  # Listen for key presses

Instructions:

* Use Left Arrow and Right Arrow to rotate the player.
* Use Up Arrow to increase speed and Down Arrow to decrease speed.
* The player can move around the game area but will turn upon hitting boundaries.

**Link to assignment folder with code and resources to test:** <https://1drv.ms/f/s!AkUk12H_fKWLjM56LLY_kpm_72l4SQ?e=A1weqq>

## Question 3

(Codex, 2023), (Fincher, 2023)

# Define Ally class (inherits from Sprite)

class Ally(Sprite):

    def \_\_init\_\_(self, shape\_type, color, initial\_x, initial\_y):

        super().\_\_init\_\_(shape\_type, color, initial\_x, initial\_y)

        self.speed = 8  # Faster than player

        self.setheading(random.randint(0, 360))  # Random direction

# Define Particle class (used for explosion effects)

class Particle(Sprite):

    def \_\_init\_\_(self, shape\_type, color, initial\_x, initial\_y):

        super().\_\_init\_\_(shape\_type, color, initial\_x, initial\_y)

        self.shapesize(stretch\_wid=0.1, stretch\_len=0.1)  # Small size

        self.goto(-1000, -1000)  # Hide off-screen initially

        self.frame = 0  # Frame counter for explosion effect

    def explode(self, initial\_x, initial\_y):

        self.goto(initial\_x, initial\_y)  # Start explosion at given position

        self.setheading(random.randint(0, 360))  # Random explosion direction

        self.frame = 1  # Start the explosion animation

    def move(self):

        if self.frame > 0:

            self.fd(10)  # Move outward

            self.frame += 1  # Increment frame

        if self.frame > 15:  # End explosion after 15 frames

            self.frame = 0

            self.goto(-1000, -1000)  # Hide off-screen

# Define Game class to manage game state and logic

class Game():

    def \_\_init\_\_(self):

        self.level = 1

        self.score = 0

        self.state = "playing"

        self.pen = turtle.Turtle()  # For displaying game info

        self.lives = 3

    def draw\_border(self):

        # Draw the game boundary

        self.pen.speed(0)

        self.pen.color("white")

        self.pen.pensize(3)

        self.pen.penup()

        self.pen.goto(-300, 300)

        self.pen.pendown()

        for side in range(4):

            self.pen.fd(600)

            self.pen.rt(90)

        self.pen.penup()

        self.pen.ht()

    def show\_status(self):

        # Display the current score

        self.pen.undo()  # Clear previous message

        msg = f"Score: {self.score}"

        self.pen.goto(-300, 310)

        self.pen.write(msg, font=("Arial", 16, "normal"))

# Create game object and initialize

game = Game()

game.draw\_border()  # Draw the game border

game.show\_status()  # Show the initial score

# Create player, enemy\_fleet, ally\_group, and particles

player = Player("triangle", "white", 0, 0)

missile = Missile("triangle", "yellow", 0, 0)

enemy\_fleet = [Enemy("circle", "red", -100, 0) for \_ in range(6)]

ally\_group = [Ally("square", "blue", 100, 0) for \_ in range(6)]

particles = [Particle("circle", "orange", 0, 0) for \_ in range(20)]

# Main game loop

while True:

    turtle.update()  # Update the screen

    time.sleep(0.02)  # Control game speed

    player.move()

    missile.move()

    for enemy in enemy\_fleet:

        enemy.move()

        if player.is\_collision(enemy):

            os.system("afplay explosion.mp3&")  # Play explosion sound

            enemy.goto(random.randint(-250, 250), random.randint(-250, 250))

            game.score -= 100  # Decrease score

            game.show\_status()

        if missile.is\_collision(enemy):

            os.system("afplay explosion.mp3&")

            enemy.goto(random.randint(-250, 250), random.randint(-250, 250))

            missile.status = "ready"

            game.score += 100  # Increase score

            game.show\_status()

            for particle in particles:

                particle.explode(missile.xcor(), missile.ycor())

    for ally in ally\_group:

        ally.move()

        if missile.is\_collision(ally):

            os.system("afplay explosion.mp3&")

            ally.goto(random.randint(-250, 250), random.randint(-250, 250))

            missile.status = "ready"

            game.score -= 50  # Decrease score

            game.show\_status()

    for particle in particles:

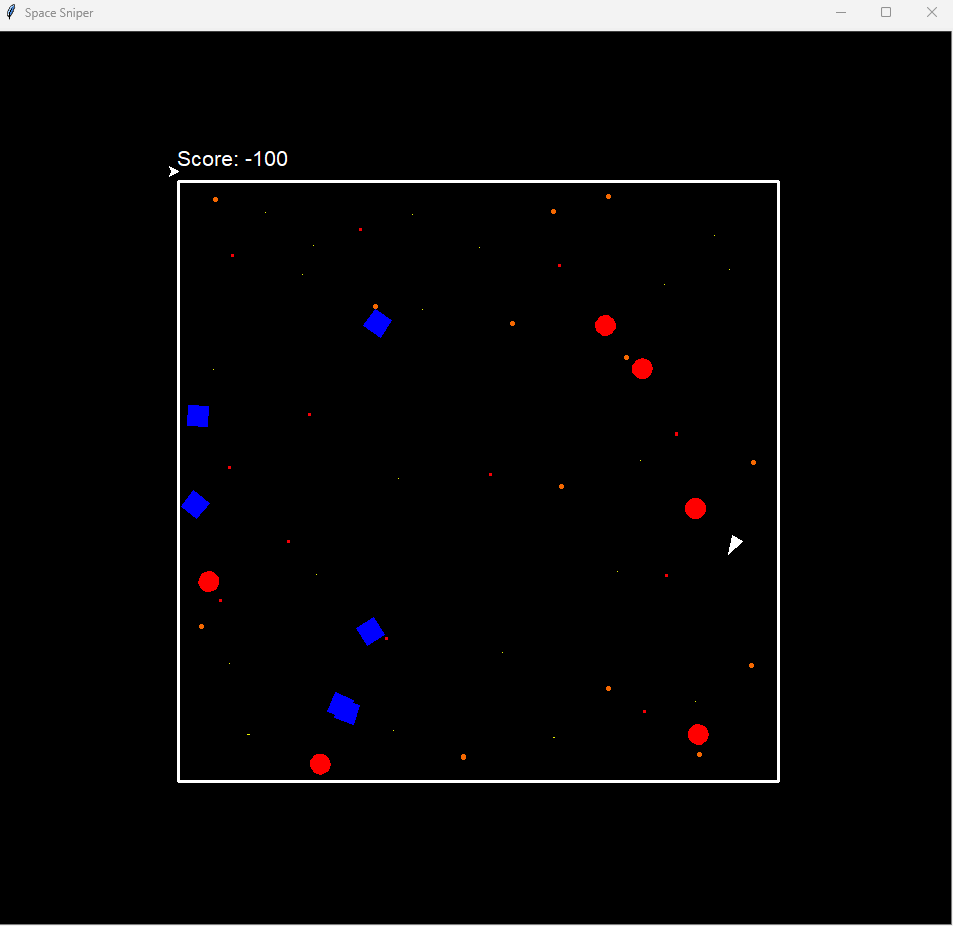
        particle.move()

delay = input("Please press enter to finish. >")

Instructions:

* Use Left Arrow and Right Arrow to rotate the player.
* Use Up Arrow to increase speed and Down Arrow to decrease speed.
* The player can move around the game area but will turn upon hitting boundaries.

**Screenshot of game:**



 - player

 - ally

 - enemy

## Question 4

### 4.1.

The Project Management Triangle consists of three main constraints which are Scope, Time, and Cost. For *Space Sniper*, these aspects are balanced as follows (Team Asana, 2024):

Scope (Radosław, 2024):

* Core scope – Single-player keyboard-controlled retro space shooter with enemies, allies and missiles. Retro aesthetic with pixel-based graphics, starry backgrounds and simple animations. Limited but exciting interactions: move, fire missiles and avoid enemies. Player progression: points system and life management. The end goal is similar to the arcade game, survive waves of enemies to get a high score.
* Scope management – It's small but polished, focusing on core gameplay rather than advanced features like a multiplayer mode. The scope of future options will include additional levels or maps and more power-ups and different enemy types.

Time:

Estimated Time to Completion:

* Design Phase will be 2-3 weeks; where this time will be used to conceptualize the game, including defining mechanics and creating wireframes.
* Development Phase will be 6 weeks, which will be used to cover programming, graphics, and sound integration.
* Testing and Debugging will be 3-4 weeks to make sure that the game is free from critical bugs.
* Total Time: approximately 12 weeks from concept to deployment.
* Time is kept relatively short to ensure quick market entry and capitalize on the rising retro gaming trend.

Cost:

Breakdown of Costs:

* Development:
  + 1 Developer working part-time as consultant which will cost R195/hour, for 30 hours will be R5 850 (Pay scale, n.d.)
* Art and Sound Assets:
  + Outsourcing retro pixel art for background of the different levels: R1 725.
* Marketing:
  + Social media campaigns, early-access demos, and online ads: R2 000 (King, 2024)
* Infrastructure:
  + Hosting for online game stores and digital distribution: R1 200 per month (Jorri, 2023).

The scope is intentionally limited to ensure high quality gameplay and fast time to market. Time is managed by setting clear milestones (design, development, testing) over 3 months to control costs and meet deadlines. Costs are optimized by limiting advanced features and using part-time developers to ensure the game stays within budget while still being marketable.

### 4.2.

**Case 1: Geometry Wars 3: Dimensions** (Otero, 2014)

Geometry Wars 3 is an arcade-style top-down shooter with simple geometric graphics, fast-paced gameplay, and a focus on score.

Similarities - Space Sniper and Geometry Wars are both retro-style arcade shooters with an emphasis on high-score competition and minimal story elements. Both games use simple control schemes (keyboard for Space Sniper, gamepad/keyboard for Geometry Wars).

Monetisation Strategies:

Paid Game - Sold for around $15 on platforms like Steam, PlayStation Store, and Xbox Live.

Seasonal Discounts - Regular discounts increase player numbers and sales.

DLC (Downloadable Content) - Additional levels and challenges sell for around $5 each, extending the life of the game.

Outcome - The game generated significant revenue from a combination of sales and DLC, grossing over $1 million in less than a year.

Lessons for Space Sniper - Offering a paid version with seasonal discounts can attract early adopters. Future potential revenue could come from DLC with new levels or power-ups.

**Case 2: Enter the Gungeon** (Stream, 2016)

Enter the Gungeon is a dungeon crawler shooter with pixel art graphics and challenging gameplay. It gained popularity for its retro aesthetic and high replay value.

Similarities- Both games feature pixel-based graphics and challenging, fast-paced shooting mechanics. Both emphasize survival, reflex-based gameplay, and scoring mechanics.

Monetisation Strategies:

Paid Model - will be released on PC and consoles for around $15.

Merchandise - developers have partnered with online stores to sell themed merchandise (t-shirts, plush toys, etc.)

Expansion Packs: To keep the community engaged, free and paid expansions have been released for.

Streaming and speed running community - The game benefited from streaming platforms such as Twitch, which further promoted the game.

Outcome - Enter the Gungeon grossed over $3 million within a year of release and continued to generate revenue through expansions and merchandise.

Lessons for Space Sniper - The game's simplicity and polish make it work well for a paid release. Collaborating with the streaming community can increase visibility and sales. Merchandise or themed items may provide additional sales opportunities.

# References

Berrena, A., 2023. *6 Game Monetization Models.* [Online]   
Available at: https://www.playwire.com/blog/6-game-monetization-models  
[Accessed 1 October 2024].

Jorri, T., 2023. *Your Comprehensive Guide To Mobile Game Server Costs.* [Online]   
Available at: https://metaplay.io/blog/mobile-game-server-costs  
[Accessed 6 October 2024].

King, M., 2024. *Mobile Game Marketing Costs.* [Online]   
Available at: https://www.businessofapps.com/marketplace/mobile-game-marketing/research/mobile-game-marketing-costs/  
[Accessed 6 October 2024].

Moraneus, 2024. *Exploring Game Development with Pygame: A Beginner’s Guide.* [Online]   
Available at: https://medium.com/@moraneus/exploring-game-development-with-pygame-a-beginners-guide-58115caf9553  
[Accessed 1 October 2024].

Otero, J., 2014. *Geometry Wars 3: Dimensions Review.* [Online]   
Available at: https://www.ign.com/articles/2014/12/02/geometry-wars-3-dimensions-review  
[Accessed 20 October 2024].

Pay scale, n.d. *Hourly Rate for Industry: Consulting.* [Online]   
Available at: https://www.payscale.com/research/ZA/Industry=Consulting/Hourly\_Rate  
[Accessed 6 October 2024].

Python, n.d. *turtle — Turtle graphics¶.* [Online]   
Available at: https://docs.python.org/3/library/turtle.html  
[Accessed 1 October 2024].

Radosław, K., 2024. *The Importance of Scope in Game Development: How to Control It and Avoid Common Pitfalls.* [Online]   
Available at: https://medium.com/@kaskiewicz.radoslaw/the-importance-of-scope-in-game-development-how-to-control-it-and-avoid-common-pitfalls-5f40b5194c29#:~:text=In%20simple%20terms%2C%20scope%20refers,audio%20design%2C%20and%20technical%20requirements.  
[Accessed 6 October 2024].

Stream, 2016. *Enter the Gungeon.* [Online]   
Available at: https://store.steampowered.com/app/311690/Enter\_the\_Gungeon/  
[Accessed 20 October 2024].

Team Asana, 2024. *What is the project management triangle and how can it help your team?.* [Online]   
Available at: https://asana.com/resources/project-management-triangle  
[Accessed 5 October 2024].