Turtle Space Shooter

1. Project Overview

Turtle Space Shooter is a simple arcade game built with Python Turtle.

The player controls a spaceship at the bottom of the screen. Enemy ships appear from the top and move downward. The player shoots them using the Space key.

The goal is to destroy as many enemies as possible and get the highest score.

2. Features

- Player spaceship
- Left and right movement
- Shooting with Space
- Multiple enemies
- Random enemy positions
- Bullet movement
- Collision detection
- Score system
- Lives
- Increasing difficulty
- Game over

3. Technologies

- Python 3
- Turtle
- Random
- Functions

- Lists
- Loops
- Conditions
- Keyboard events
- Coordinates
- Timers

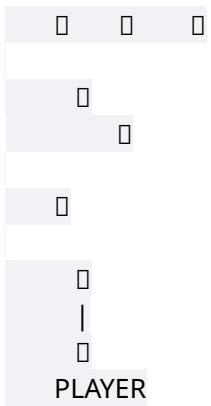
No external packages are required.

4. Folder Structure

```
turtle_space_shooter/
□□□ main.py
□□□ game.py
□□□ README.md
```

5. How the Game Works

The player controls a spaceship at the bottom.



The player uses:

 Move left
 Move right
SPACE Shoot

Enemy ships continuously move downward.

When a bullet touches an enemy, the enemy disappears and the score increases.

If an enemy reaches the bottom, the player loses a life.

The game ends when all lives are lost.

6. Complete Game Code

game.py

```
import random
import turtle


class SpaceShooter:
    def __init__(self):
        self.screen = turtle.Screen()
        self.screen.title("Turtle Space Shooter")
        self.screen.setup(width=800, height=700)
        self.screen.bgcolor("black")
        self.screen.tracer(0)

        self.score = 0
        self.lives = 3
        self.game_over = False

        self.player_speed = 30
        self.bullet_speed = 20
```

```
        self.enemy_speed = 4

        self.player = None
        self.bullets = []
        self.enemies = []

        self.score_writer = turtle.Turtle()
        self.score_writer.hideturtle()
        self.score_writer.color("white")
        self.score_writer.penup()
        self.score_writer.goto(-370, 310)

        self.create_stars()
        self.create_player()
        self.create_enemies()
        self.setup_controls()
        self.update_score()

    def create_stars(self):
        for _ in range(40):
            star = turtle.Turtle()
            star.shape("circle")
            star.color("white")
            star.shapesize(0.08)
            star.penup()

            star.goto(
                random.randint(-380, 380),
                random.randint(-300, 300)
            )

    def create_player(self):
        self.player = turtle.Turtle()
        self.player.shape("triangle")
```

```
        self.player.color("cyan")
        self.player.penup()
        self.player.setheading(90)
        self.player.goto(0, -270)

    def create_enemies(self):
        colors = ["red", "orange", "yellow", "purple"]

        for _ in range(5):
            enemy = turtle.Turtle()
            enemy.shape("circle")
            enemy.color(random.choice(colors))
            enemy.penup()

            enemy.goto(
                random.randint(-350, 350),
                random.randint(100, 320)
            )

            self.enemies.append(enemy)

    def setup_controls(self):
        self.screen.listen()

        self.screen.onkeypress(
            self.move_left,
            "Left"
        )

        self.screen.onkeypress(
            self.move_right,
            "Right"
        )
```

```
        self.screen.onkeypress(
            self.shoot,
            "space"
        )

    def move_left(self):
        if self.game_over:
            return

        new_x = self.player.xcor() - self.player_speed

        if new_x > -360:
            self.player.setx(new_x)

    def move_right(self):
        if self.game_over:
            return

        new_x = self.player.xcor() + self.player_speed

        if new_x < 360:
            self.player.setx(new_x)

    def shoot(self):
        if self.game_over:
            return

        bullet = turtle.Turtle()
        bullet.shape("square")
        bullet.color("yellow")
        bullet.shapesize(
            stretch_wid=0.5,
            stretch_len=0.2
        )
```

```
        bullet.penup()

        bullet.goto(
            self.player.xcor(),
            self.player.ycor() + 20
        )

        self.bullets.append(bullet)

    def move_bullets(self):
        for bullet in self.bullets[:]:
            bullet.sety(
                bullet.ycor() + self.bullet_speed
            )

            if bullet.ycor() > 350:
                bullet.hideturtle()
                self.bullets.remove(bullet)

    def move_enemies(self):
        for enemy in self.enemies:
            enemy.sety(
                enemy.ycor() - self.enemy_speed
            )

            if enemy.ycor() < -340:
                enemy.goto(
                    random.randint(-350, 350),
                    random.randint(300, 450)
                )

                self.lives -= 1
                self.update_score()
```

```
            if self.lives <= 0:
                self.end_game()

    def check_collisions(self):
        for bullet in self.bullets[:]:
            for enemy in self.enemies:

                if bullet.distance(enemy) < 25:
                    bullet.hideturtle()

                    if bullet in self.bullets:
                        self.bullets.remove(bullet)

                    enemy.goto(
                        random.randint(-350, 350),
                        random.randint(300, 450)
                    )

                    self.score += 1
                    self.update_score()

                    break

    def update_score(self):
        self.score_writer.clear()

        self.score_writer.write(
            f"Score: {self.score}    Lives: {self.lives}",
            font=("Arial", 16, "bold")
        )

    def end_game(self):
        if self.game_over:
            return
```

```
        self.game_over = True

        message = turtle.Turtle()
        message.hideturtle()
        message.color("white")
        message.penup()

        message.goto(0, 60)

        message.write(
            "GAME OVER",
            align="center",
            font=("Arial", 32, "bold")
        )

        message.goto(0, 10)

        message.write(
            f"Final Score: {self.score}",
            align="center",
            font=("Arial", 20, "normal")
        )

        message.goto(0, -40)

        message.write(
            "Close the window to exit",
            align="center",
            font=("Arial", 14, "normal")
        )

    def game_loop(self):
        if self.game_over:
```

```
            self.screen.update()
            return

        self.move_bullets()
        self.move_enemies()
        self.check_collisions()

        self.screen.update()

        if not self.game_over:
            self.screen.ontimer(
                self.game_loop,
                30
            )

    def start(self):
        self.game_loop()
        self.screen.mainloop()
```

7. Main Program

main.py

```
from game import SpaceShooter


def main():
    game = SpaceShooter()
    game.start()


if __name__ == "__main__":
    main()
```

8. How to Run

Open the project in VS Code.

Run:

```
python main.py
```

Use:

```
 Left Arrow     Move left

 Right Arrow    Move right

SPACE           Shoot
```

Destroy enemy ships before they reach the bottom.

9. What Students Learn

Creating Objects

```
player = turtle.Turtle()
```

Students learn how to create and control Turtle objects.

Keyboard Events

```
screen.onkeypress(
    move_left,
    "Left"
)
```

Keyboard keys can be connected to functions using Turtle's keyboard event system. The screen needs focus through `listen()` to receive these events.

Lists

```
self.bullets = []
self.enemies = []
```

Lists allow the game to keep track of multiple bullets and enemies.

Random Numbers

```
random.randint(-350, 350)
```

This gives enemies different positions.

Collision Detection

```
if bullet.distance(enemy) < 25:
```

The game checks whether a bullet is close enough to an enemy to count as a hit.

Game Loop

```
self.screen.ontimer(
    self.game_loop,
    30
)
```

The game repeatedly updates bullets, enemies and collisions. Turtle's `ontimer()` schedules a function to run after a specified number of milliseconds.

10. Basic Testing

Students should check:

- Left arrow moves the spaceship.

- Right arrow moves the spaceship.
- Space creates bullets.
- Bullets move upward.
- Enemies move downward.
- Bullets destroy enemies.
- Score increases after a hit.
- Missed enemies reduce lives.
- Game ends when lives reach zero.

11. Student Challenges

Once the basic version works, give students these upgrades.

Challenge 1: More Enemies

Change:

```
for _ in range(5):
```

to:

```
for _ in range(10):
```

Challenge 2: Faster Enemies

Increase:

```
self.enemy_speed = 4
```

to:

```
self.enemy_speed = 7
```

Challenge 3: Level System

Every 10 points, increase the enemy speed.

```
10 points  Level 2
20 points  Level 3
30 points  Level 4
```

Challenge 4: Power Up

Create a special object that gives the player faster shooting.

Challenge 5: Boss Enemy

After reaching 50 points, create a large boss enemy with multiple hits required.

12. Final Student Challenge

Tell students:

> **Build your own version of the Space Shooter. Add at least 3 features that are not in the original game.**

They can add:

- Boss fights
- Power ups
- Different weapons
- More lives
- Levels
- High score
- Faster enemies

- Different enemy types
- Sound effects
- Restart option
- Special attacks

13. Project Concepts

Variables

Functions

Lists

Random Numbers

Coordinates

Keyboard Events

Multiple Objects

Collision Detection

Game Loop

Score & Lives

14. Project Organization

`main.py` starts the application.

`game.py` contains:

- Player

- Enemies
- Bullets
- Keyboard controls
- Movement
- Collision detection
- Score
- Lives
- Game loop
- Game over

Visit haas.dev for more resources and guides.