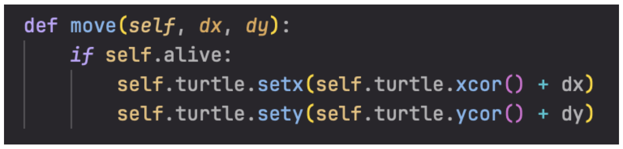
Computational Thinking Projet : Space Invaders

Complexity analysis of key components:
- Player movements logic
- Alien movements logic
- Collision detection for bullets
- Curving of bullets
- Game loop

<u>Player movements logic :</u>

Function **move** :

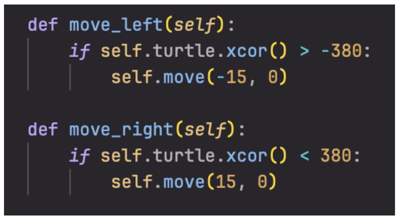


```python
def move(self, dx, dy):
    if self.alive:
        self.turtle.setx(self.turtle.xcor() + dx)
        self.turtle.sety(self.turtle.ycor() + dy)
```

Time complexity : **O(1)**, as it runs in the same time no matter the input size.
- "if" loop executes in constant time
- self.turtle.xcor() and self.turtle.ycor() retrieve the coordinates of x and y in constant time.
- Adding dx and dy is in constant time.
- self.turtle.setx() and self.turtle.sety() set the coordinates in constant time.

Space complexity : **O(1)**, as it does not any new data structures or variables proportional to input size.

Function **move_left**, **move_right** :



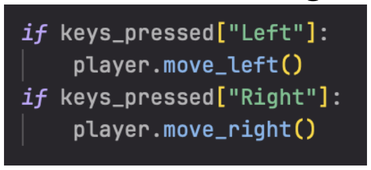
```python
def move_left(self):
    if self.turtle.xcor() > -380:
        self.move(-15, 0)

def move_right(self):
    if self.turtle.xcor() < 380:
        self.move(15, 0)
```

Used for the movements of the player, moving either right or left.
Time complexity : **O(1)** as they run in the same time no matter the input size.
- "if" condition executes in constant time
- self.turtle.xcor() retrieves the x coordinate in constant time.
- The comparaison (> -380, < 380) is a constant time operation.
- Call upon the move function, a O(1) function.

Space complexity : **O(1)**, as they don't add any new data structures or variables proportional to input size.

Part of function **start_game**, **“if keys_pressed”:**



```python
if keys_pressed["Left"]:
    player.move_left()
if keys_pressed["Right"]:
    player.move_right()
```

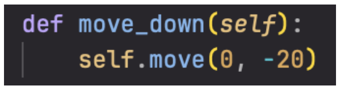
Used for the movements of the player, to execute functions move_right and move_left when keyboard keys are pressed.
Time complexity : **O(1)**

- “if” condition runs in constant time.
- A dictionnary lookup (keys_pressed) is a constant time operation.
- Calls upon functions move_right and move_left, both of O(1) complexity.
- Calls, through move_right and move_left, the function move, which runs in O(1) complexity.

Space complexity : **O(1)**, as accessing a dictionnary or other operations doesn’t add new data or use additional memory.

<u>Alien movements logic :</u>

Function **move_down** :



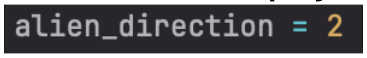
```python
def move_down(self):
    self.move(0, -20)
```

Used for the movements of aliens, moving from the top down.
Time complexity : **O(1)** as it runs in the same time no matter the input size.

- “If” loop executes in constant time
- Call upon the move function, a O(1) function.

Space complexity : **O(1)**, as it doesn’t any new data structures or variables proportional to input size.

Line of function **display_message**, **“alien direction = 2”**



```python
alien_direction = 2
```

This line determines the horizontal direction of the aliens.
When the value is positive (2), aliens will move right, when the value is negative (-2), aliens will move left.
The value changes when the alien reach the edge of the game display.

<u>Collision detection for bullets :</u>

Function **is_collision** :

```python
def is_collision(obj1, obj2):
    if obj1.alive and obj2.alive:
        distance = math.hypot(
            obj1.turtle.xcor() - obj2.turtle.xcor(),
            obj1.turtle.ycor() - obj2.turtle.ycor()
        )
        return distance < 20  # Collision threshold
    return False
```

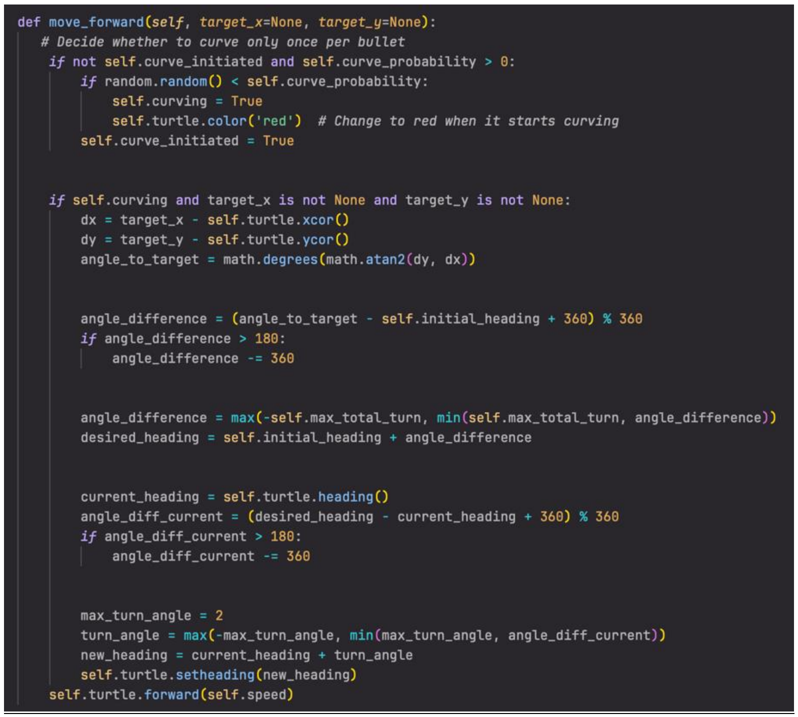
Checks if an obj1 and an obj2 (bullet and player) collide.
Time complexity : **O(1)**, as it runs in the same time no matter the input size.

- "if" condition runs in constant time.
- Checking the "alive" property of obj1 and obj2 is a constant operation.
- turtle.xcor() and turtle.ycor() retrieve the x and y coordinates which is a constant operation.
- The comparison (<20) is a constant operation.

Space complexity : **O(1)**, as it does not any new data structures or variables proportional to input size.

<u>Curving of bullets :</u>

Function **move_forward** :



```python
def move_forward(self, target_x=None, target_y=None):
    # Decide whether to curve only once per bullet
    if not self.curve_initiated and self.curve_probability > 0:
        if random.random() < self.curve_probability:
            self.curving = True
            self.turtle.color('red')  # Change to red when it starts curving
        self.curve_initiated = True


    if self.curving and target_x is not None and target_y is not None:
        dx = target_x - self.turtle.xcor()
        dy = target_y - self.turtle.ycor()
        angle_to_target = math.degrees(math.atan2(dy, dx))


        angle_difference = (angle_to_target - self.initial_heading + 360) % 360
        if angle_difference > 180:
            angle_difference -= 360


        angle_difference = max(-self.max_total_turn, min(self.max_total_turn, angle_difference))
        desired_heading = self.initial_heading + angle_difference


        current_heading = self.turtle.heading()
        angle_diff_current = (desired_heading - current_heading + 360) % 360
        if angle_diff_current > 180:
            angle_diff_current -= 360


        max_turn_angle = 2
        turn_angle = max(-max_turn_angle, min(max_turn_angle, angle_diff_current))
        new_heading = current_heading + turn_angle
        self.turtle.setheading(new_heading)
    self.turtle.forward(self.speed)
```

Decides whether to curve a bullet and sets instructions for curving.
Time complexity : **O(1)**,

- For the decision on curving :
    - “if” condition runs in constant time.
    - Comparaison between a random value and self.curve_probability is a constant time operation.
- For the logic of curving :
    - “if” condition runs in constant time.
    - All operations are constant : substraction, comparaison, …

Space complexity : **O(1)**, as it does not any new data structures or variables proportional to input size.

<u>Game loop :</u>

Function **start_game** :
The start_game function drives the main game loop. Its complexity is determined by how it processes player input, updates game objects, checks collisions, and handles game states within the while running loop.

Time complexity analysis :

- Input of player : « if keys_pressed” has O(1) complexity as seen above.
- Logic of power-ups :
    - checking the conditions for a power-up to spawn and creating a new one takes a constant amount of time. (O(1))
    - checking collision detection and expiration for all active power-ups has a complexity of O(P), with P as the number of active power-ups.
    - Overall : O(P)
- Movement of bullets and collision detection :
    - For player bullets : Iterating over player_bullets to move them and see if collisions with aliens or bosses takes O(Bp * (A + 1)), where Bp is the number of player bullets, A is the number of aliens, and "+1" accounts for bosses.
    - For alien bullets : Iterating over alien_bullets takes O(Ba) to move them and make sure they are paired with players, where Ba is the number of alien bullets.
    - Overall : O(Bp * (A + 1) + Ba)
- Alien or boss movement :
    - For boss : Moving it and checking its position are constant-time operations, so O(1).

    - For aliens : It takes O(A) to check over all the aliens to update their position or to move them down when they reach an edge of the game display, with A the number of aliens.
    - Overall : O(A)
- Logic of alien and boss firing :
    - Checking firing conditions for the boss is O(1).
    - Iterating over all aliens to decide whether they fire a bullet takes O(A).
    - Overall : O(A)
- Waves complexity :
    - Checking if all aliens or boss are destroyed and initiating a new wave takes constant time O(1).
- Frame update :
    - Sets a fixed time for frame updates. This operation takes is of O(1) complexity.

Overall Time complexity :
The dominant complexity term is the movement of bullets and collision detection (O(Bp * (A + 1))). Considering furthermore the logic of power-ups, the alien or boss movement, the logic of alien and boss firing, the total complexity is :
O(Bp * (A + 1) + Ba + A + P), with Bp the number of player bullets, Ba the number of alien bullets, A the number of aliens, P the number of active power-ups.

Overall Space complexity :
The space complexity of the game loop depends on the number of active objects in the game at any given time. These include player bullets, alien bullets, aliens, and power-ups. The memory used by the function is proportional to the total number of these objects, making the space complexity **O(Bp + Ba + A + P).**