CPEN 333 Final Project : Part I (Multithreaded Game)

Group G6 : Muntakim Rahman and Tomaz Zlindra

December 6th, 2024

1 Requirements and Constraints

We were provided the template with skeleton code (i.e. classes, methods with docstrings) for the implementation of the Snake Game. The key design structure entailed the following requirement:

Use `Queue.queue` module to ensure multi-producer, multi-consumer achieves synchronization between threads (i.e. add `dict` items to the queue for "`game_over`", "`score`", "`prey`", "`move`").

2 Implementation

We were provided fully functional `Gui`, `QueueHandler` classes with a `gameQueue` instance of the standard `Python Queue`' class. Our task was to implement the following methods of the `Game` class.

- `superloop()`
- `move()`
- `calculateNewCoordinates()`
- `isGameOver(snakeCoordinates:tuple)`
- `createNewPrey()`

2.1 UML Relationships

We utilized the template data fields and methods to program a responsive and thread-safe Snake Game, as shown in the UML Relationships in Figure 1.

These enabled the program to update the game state via the `Game` class and display this to the user via the `Tkinter` widgets in the `Gui` class.

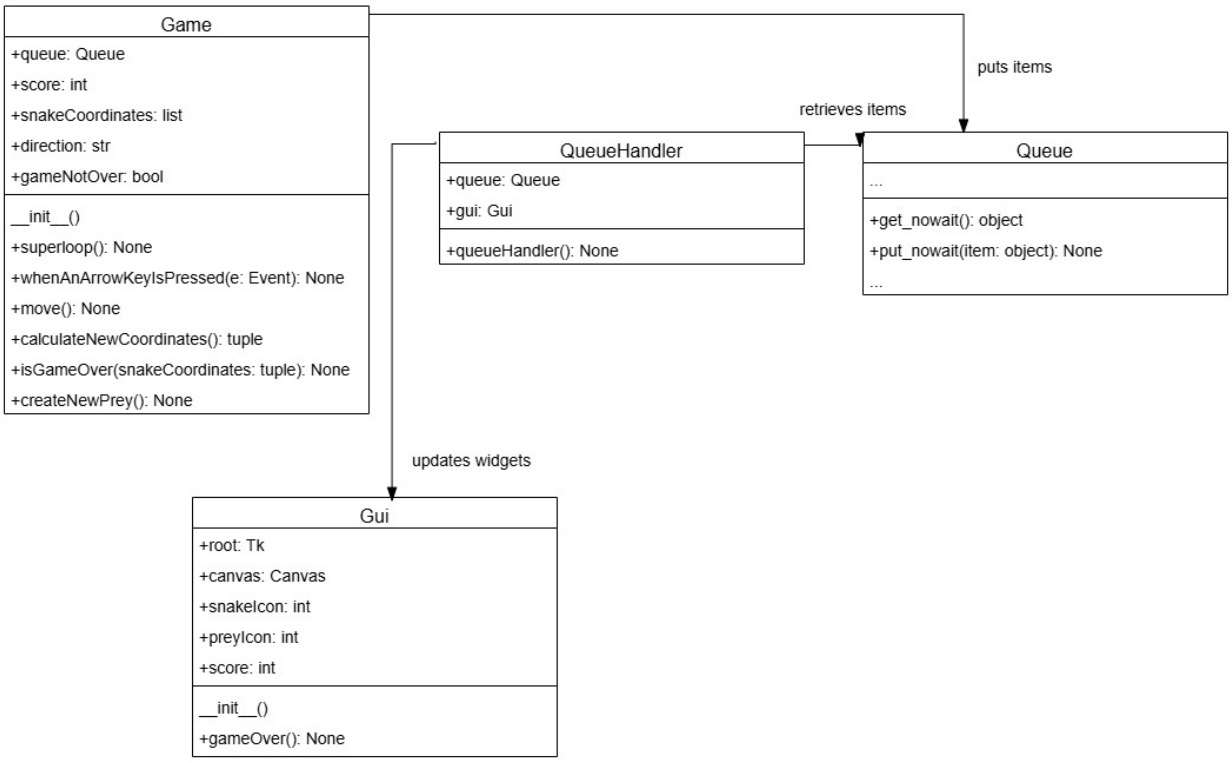

Figure 1: Part 1 - UML Relationships

2.2 Gameplay

The game is played with the `superloop()` method in a `daemonic thread`, until the `game.gameNotOver` data field is read as `False`. A conditional loop in the `superloop()` method calls the `move()` method every `SPEED` s to perform the following actions:

1. Determine the new head coordinates of the snake (i.e. based on current direction). (See `calculateNewCoordinates()`.)
2. If the prey has been captured: (See `isCaptured(snakeCoordinates: tuple, preyCoordinates: list)`.)
   (a) Append the head to the `game.snakeCoordinates` data field (i.e. increase its length by `SNAKE_ICON_WIDTH`).
   (b) Increment the score and **put this in the** `gameQueue`.
   (c) Randomly pick a valid (x, y) coordinate pair for the new prey. **Put this in the** `gameQueue`**.** (See `createNewPrey()`.)
3. If the prey has not been captured :
   (a) Shift the the `game.snakeCoordinates` data field forward by one item (i.e. to simulate movement).
4. If the snake has hit itself / the wall : update the `game.gameNotOver` data field to be `False`. **Put this in the** `gameQueue`**.** (See `isGameOver(self, snakeCoordinates: tuple)`.)
5. **Put the new coordinates list (i.e.** `game.snakeCoordinates` **data field) in the** `gameQueue`**.**

Note that items are put in the `gameQueue` **with the** `put_nowait(item)` **method from the** `Queue` **class since it is non-blocking.**

Let's look at a few of the implemented methods in detail.

2.2.1 `calculateNewCoordinates()`

A conditional statement method reads the current value of the `game.direction` data field to shift the head coordinates in the respective direction. This is computed as an addition/substraction of the relevant x or y coordinate by `SNAKE_ICON_WIDTH`.

2.2.2 `createNewPrey()`

We used the `randint()` function from the `random` library to generate an integer for the (x, y) coordinate pair. This follows the calculation in the provided docstring to consider coordinates a `THRESHOLD` margin away from the edge of the `Tkinter Canvas`. We subtracted/added half of the `PREY_ICON_WIDTH` in either direction and put these edge coordinates into the `gameQueue`.

Note that the new prey can spawn in the same coordinates as the snake body / head. This is by design (i.e. for simplicity) and may be considered a capture based on the capture logic below.

2.2.3 `isCaptured(snakeCoordinates: tuple, preyCoordinates: list)`

We designed the logic for capturing the prey to account for different sizes of both the prey and snake head. This was captured by the inner function in the `move` method.

We determined the edge coordinates of both the prey and the snake's head. We then checked whether any point of the snake's head fell within the prey's boundaries (and vice-versa), as illustrated in figure 2.

Note that the `Canvas.coords(gui.preyIcon)` method was used to retrieve the x0, y0, x1, y1 points.

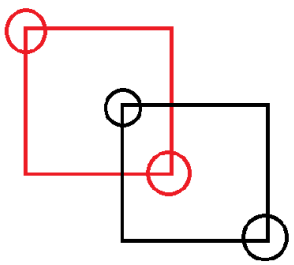

Figure 2: Prey Capture Criteria : Edge Coordinate Of Prey/Snake Must Be Contained

Since `Canvas.create_line(...)` renders a *1-D* widget for the `gui` instance, we considered a *radius* of half the `SNAKE_ICON_WIDTH` along the relevant x or y coordinate, depending on the current value of the `game.direction` data field. This is similar to the logic in the `createNewPrey()` method.

2.2.4 `isGameOver(self, snakeCoordinates: tuple)`

The game ends once either of the following has occurred:

- snake's head exits the canvas bounds
- snake's head hits its body (i.e. head and body have same (x, y) coordinate pair)