1 Introduction

Snake is a classic game where the player controls a snake on the screen. The goal is to steer the snake towards apples, and to avoid hitting either the snake or the edge of the screen.

2 Implementation

2.1 The Vector abstraction

The `Vec2` class represents a 2-dimensional vector. For our purposes, it doesn't need many of the fancier linear algebra functions.

```
\class{Vec2}{x, y}{
    self.x = x;
    self.y = y;
}


\fun{Vec2::clone}{}{
    return Vec2(self.x, self.y);
}


\fun{Vec2::set}{other}{
    self.x = other.x;
    self.y = other.y;
    return self;
}


\fun{Vec2::add}{other}{
    self.x = self.x + other.x;
    self.y = self.y + other.y;
    return self;
}
```

2.2 The Game class

The `Game` class represents the state of the game.

```
\class{Game}{canvas, w, h}{
    self.canvas = canvas;
    self.width = w;
    self.height = h;
```

```
    self.ctx = canvas.getContext("2d");
    self.prevTime = none;
    self.direction = Vec2(0 - 1, 0);
    self.nextDirection = none;
    self.bufferedDirection = none;
    self.snake = Array();
    self.moveTimer = 0;
    self.moveTime = 0.5;

    self.scale = 20;
    self.canvas.width = self.width * self.scale;
    self.canvas.height = self.height * self.scale;

    startPos := Vec2(math.floor(w / 2), math.floor(h / 2));
    self.snake.push(Vec2(startPos.x + 2, startPos.y));
    self.snake.push(Vec2(startPos.x + 1, startPos.y));
    self.snake.push(Vec2(startPos.x + 0, startPos.y));

    self.apple = none;
    self.spawnApple();

    self.lost = false;
}
```

2.3 Spawning Apples

The `spawnApple` function's responsibility is to spawn a new apple, in a random location. In a future iteration, it could be a good idea to avoid spawning the apple right at the head of the snake.

```
\fun{Game::spawnApple}{}{
    x := math.floor(math.random() * self.width);
    y := math.floor(math.random() * self.height);
    self.apple = Vec2(x, y);
}
```

2.4 Moving the snake

The `move` function moves the snake one square, according to the current direction. If the snake hits an apple, the snake grows instead of moving. If the snake hits a wall or itself, that's game over.

```
\fun{Game::move}{}{
```

```
    if self.nextDirection != none {
        self.direction = self.nextDirection;
        self.nextDirection = self.bufferedDirection;
        self.bufferedDirection = none;
    }

    if self.checkForApple() {
        return none;
    }

    index := 0;
    while index < self.snake.length - 1 {
        curr := self.snake.get(index);
        curr.set(self.snake.get(index + 1));
        index = index + 1;
    }

    head := self.snake.get(self.snake.length - 1);
    head.add(self.direction);

    self.checkForGameOver();
}
```

2.5 Checking for collision with an apple

The `checkForApple` function checks if the snake is about to move onto a square with an apple. If it is, then none of the existing snake parts are actually moved; instead, a new snake part is added to the end of the snake, and an apple is spawned in another location.

```
\fun{Game::checkForApple}{}{
    nextPos := self.snake.get(self.snake.length - 1).clone().add(self.direction);
    if nextPos.x == self.apple.x * nextPos.y == self.apple.y {
        self.snake.push(nextPos);
        self.spawnApple();
        self.moveTime = self.moveTime * 0.95;
        return true;
    }

    return false;
}
```

2.6 Checking for the lose conditions

The `checkForGameOver` function checks for the two lose conditions. The game is lost if:

- The head of the snake collides with any part of the snake body; or
- The head of the snake collides with any of the four walls of the game area.

If either of these conditions occur, the game is lost.

```
\fun{Game::checkForGameOver}{}{
    head := self.snake.get(self.snake.length - 1);
    index := 0;
    while index < self.snake.length - 1 {
        curr := self.snake.get(index);
        if (head.x == curr.x) * (head.y == curr.y) {
            self.lost = true;
            return true;
        }

        index = index + 1;
    }

    if (head.x < 0) + (head.x >= self.width) + (head.y < 0) + (head.y >= self.height) {
        self.lost = true;
        return true;
    }

    return false;
}
```

2.7 Drawing the game board

The game board has to be drawn every frame. The responsibilities of the `draw` function are:

- If the game is lost, show the game over screen. Otherwise:
- Paint over the canvas with the background color (cyan).
- Draw all the sections of the snake.
- Draw the apple.

```
\fun{Game::draw}{}{
    if self.lost {
```

```
            self.ctx.fillStyle = "black";
            self.ctx.font = "24px sans-serif";
            self.ctx.fillText("You lost!", 8, 24 + 8);
            self.ctx.font = "18px sans-serif";
            self.ctx.fillText("Your score: " + self.snake.length, 8, 24 + 32);
            return none;
        }

        self.ctx.save();
        self.ctx.scale(self.scale, self.scale);

        self.ctx.fillStyle = "cyan";
        self.ctx.fillRect(0, 0, self.width, self.height);

        self.ctx.lineWidth = 1/15;

        index := 0;
        self.ctx.fillStyle = "green";
        self.ctx.strokeStyle = "black";
        while index < self.snake.length {
            curr := self.snake.get(index);
            self.ctx.beginPath();
            self.ctx.rect(curr.x, curr.y, 1, 1);
            self.ctx.fill();
            self.ctx.stroke();
            index = index + 1;
        }

        self.ctx.fillStyle = "yellow";
        self.ctx.strokeStyle = "black";
        self.ctx.beginPath();
        self.ctx.arc(self.apple.x + 0.5, self.apple.y + 0.5, 0.3, 0, 2 * math.PI);
        self.ctx.fill();
        self.ctx.stroke();

        self.ctx.restore();
    }
```

2.8 Input handling

The `onKey` function's responsibility is to respond to key presses. In Snake, the only possible player input is to set the current move direction to any of the four cardinal directions. This controls the direction which the snake will move the next time the snake moves.

```
\fun{Game::onKey}{key}{
    dir := none;
    if (key == "ArrowUp") + (key == "KeyW") {
        dir = Vec2(0, 0 - 1);
    } else if (key == "ArrowLeft") + (key == "KeyA") {
        dir = Vec2(0 - 1, 0);
    } else if (key == "ArrowDown") + (key == "KeyS") {
        dir = Vec2(0, 1);
    } else if (key == "ArrowRight") + (key == "KeyD") {
        dir = Vec2(1, 0);
    }

    if dir != none {
        if self.nextDirection == none {
            self.nextDirection = dir;
        } else if self.bufferedDirection == none {
            self.bufferedDirection = dir;
        }
    }
}
```

2.9 The game loop

The `update` function is called in a loop, with the current time (in seconds) as its only paramater. It calculates a delta time since the last frame, decides whether to move or not based on a timer, and draws the game board.

```
\fun{Game::update}{t}{
    if self.lost {
        return none;
    }

    if self.prevTime == none {
        self.prevTime = t;
        return none;
    }

    dt := t - self.prevTime;
    self.prevTime = t;

    self.moveTimer = self.moveTimer + dt;
    while self.moveTimer > self.moveTime {
        self.move();
        self.moveTimer = self.moveTimer - self.moveTime;
    }
```

```
    self.draw();
}
```