Snake Game part 7

Step 7: Snake Collisions

Now, we are almost there. The game is playable, except that the snake doesn't get out if it collides with the walls or itself. Let's add a method to detect a snake crash, and end the game. The method will look like this:

```
boolean collide() {
  for (PVector section : tail) {
    if (snake.x == section.x && snake.y == section.y) {
      return true;
    }
  }
  return false;
}
```

This method will return true if it crashes, and false if it doesn't crash.

Then, in the draw method, we can check for collisions. If we find any, we'll start a new game.

```
  if (collide()) {
    newGame();
  }
```

We'll also update our `newGame()` method to reset the tail. The `.clear()` method erases all of the values in an `ArrayList`, so `tail.clear()` will reset our tail list. Then, we set the `tailLength` back to 0. Otherwise, it will automatically add our current position into the tail list, until it is the size it was in the previous game.

```
void newGame() {
  snake = new PVector(width/2, height/2);
  snakeSize = gridSize;
  direction(1, 0);
  newFood();
  tail.clear();
  tailLength = 0;
}
```

Our code will now look like this:

```
int gridSize = 20;

PVector snake;
int snakeSize;
int xSpeed;
int ySpeed;
int tailLength;
ArrayList<PVector> tail = new ArrayList();
```

```
PVector food;

void setup() {
  size(600, 600);
  frameRate(10);
  newGame();
}

void draw() {
  background(0);
  showFood();
  eatFood(food);
  moveTail();
  moveSnake();
  showSnake();
  showTail();
  if (collide()) {
    newGame();
  }
}

void newGame() {
  snake = new PVector(width/2, height/2);
  snakeSize = gridSize;
  direction(1, 0);
  newFood();
  tail.clear();
  tailLength = 0;
}

void showSnake() {
  fill(100, 215, 0);
  square(snake.x, snake.y, snakeSize);
}

void moveSnake() {
  snake.x += xSpeed;
  snake.y += ySpeed;

  // This keeps the snake on the board
  snake.x = constrain(snake.x, 0, width-gridSize);
  snake.y = constrain(snake.y, 0, height-gridSize);
}

void moveTail() {
  if (tailLength > 0) {
    if (tailLength == tail.size() && !tail.isEmpty()) {
      tail.remove(0);
    }
    tail.add(new PVector(snake.x, snake.y));
```

```
    tail.add(new PVector(snake.x, snake.y));
  }
}

void showTail() {
  for (PVector section : tail) {
    square(section.x, section.y, gridSize);
  }
}

void direction(int x, int y) {
  xSpeed = x * gridSize;
  ySpeed = y * gridSize;
}

void keyPressed() {
  if (keyCode == UP) {
    direction(0, -1);
  } else if (keyCode == DOWN) {
    direction(0, 1);
  } else if (keyCode == RIGHT) {
    direction(1, 0);
  } else if (keyCode == LEFT) {
    direction(-1, 0);
  }
}

void newFood() {
  int columns = width/gridSize;
  int rows = height/gridSize;
  int randomColumn = (int)random(columns) * gridSize;
  int randomRow = (int)random(rows) * gridSize;
  food = new PVector(randomColumn, randomRow);
}

void showFood() {
  fill(255, 0, 100);
  square(food.x, food.y, gridSize);
}

void eatFood(PVector food) {
  if (snake.x == food.x && snake.y == food.y) {
    tailLength++;
    newFood();
  }
}

boolean collide() {
  for (PVector section : tail) {
    if (snake.x == section.x && snake.y == section.y) {
      return true;
    }
```

```
    }
  }
  return false;
}
```

Challenges

1. Add obstacles that the snake needs to avoid in middle of the board.