# Practical Test 1

## Instructions

1. This test is covers class design & implementation, and object composition.
2. You will need the Processing project PractTest1 for this test.
3. You should save your work regularly as you progress through the test.
4. Ensure that both the sketch window and the console output is clearly visible when you run the program.
5. By the end of the test, you should have *two* Processing projects to submit. Their names should be **PracTest1\_PartA** and **PracTest1\_PartB**.

## Preparation

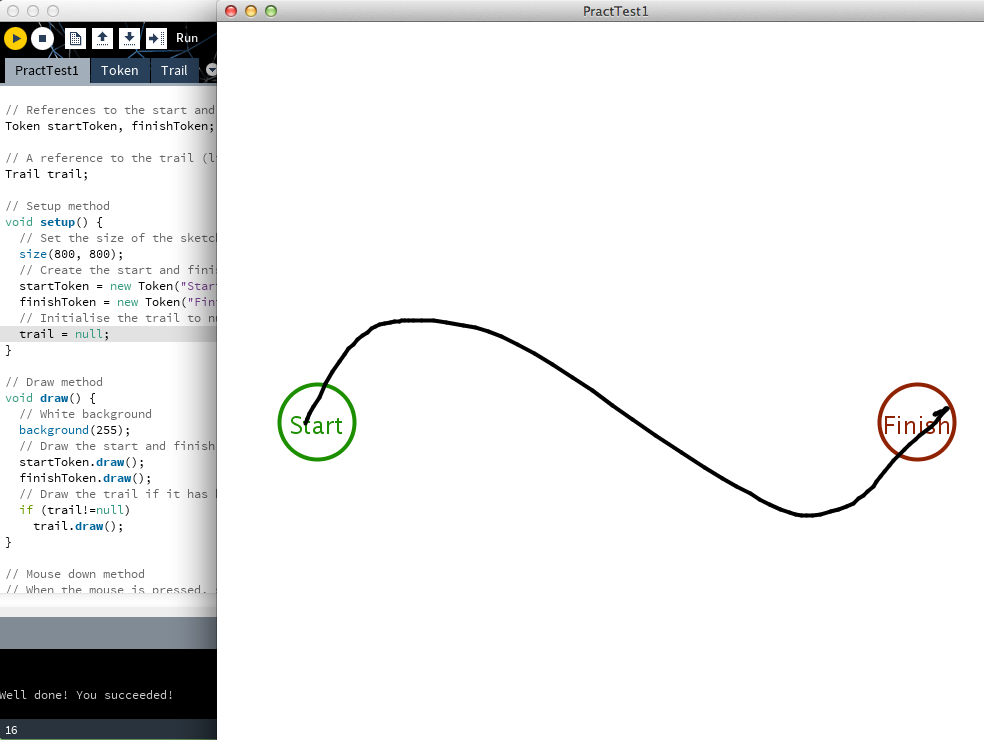
The PractTest1 project is a simple game written in Processing in which the objective is to use the mouse to draw a line (called a “trail”) from the start position to the end position.

The rules of the game are as follows:

A trail starts whenever you press the mouse button and the mouse pointer is over the start position. No trail starts if you click the mouse button anywhere else. The trail is drawn as you move the mouse while holding the button down. A trail ends whenever you release the mouse button. If the mouse button is released while the mouse pointer is over the finish position, you win; otherwise you lose.

The outcome of the game (win or lose) is printed to the console every time the mouse button is released and a trail is on the screen. At that point the trail is deleted and new game begins.

A screenshot of a successful game is shown below:



**Please test the program now to ensure that everything works as described above.**

## Program Overview

The program currently has two classes and main file with the setup()/draw() methods. An overview of these parts of the program now follows.

**The Token Class**

This class is used to represent the “start” and “finish” positions. The properties of the class are the text displayed along with various appearance properties such as sizes and colours.

A constructor initializes all of the properties.

There are two methods in this class: draw(), which draws a token, and pointInToken(), which returns true if an arbitrary point (*x,y*) lies inside the area of the token.

**The Trail Class**

This class represents the trail drawn by the user as they play the game. The main property is an ArrayList of PVectors called points, which stores the points that the user has drawn on the screen; the remaining two properties are concerned with the appearance of the trail.

The constructor initializes the arraylist and the rest of the properties.

There are two methods in this class: draw(), which draws the trail; and addPoint(), which adds a new point to the trail.

**The main tab with setup()/draw()**

This part of the program is where the main logic of the game is implemented.

Two Token objects and a Trail object are declared. Inside setup(), the tokens are created, but trail is set to null. Inside draw(), everything that is not null is drawn.

The trail object is managed inside the mouse event functions following the draw() method. Essentially, if the user presses the mouse over the start token, a new trail is created; as the user drags the mouse, points are added to the trail; and after the user releases the mouse, the game state (success or failure) is printed out and trail is set back to null.

**Please study the code now and ensure that you understand how the program works.**

## Tasks

**The test is composed of two parts worth 20 marks in total.** Both parts are worth 10 marks each. Success in Part 2 is not possible until you have Part 1 working properly.

**Part A [10 Marks]**

***Ensure that your program is renamed to PracTest1\_PartA****.*

Create a new class called Obstacle. An obstacle is a rectangle that the user must navigate their trail around when they play the game.

An Obstacle has the following properties:

* a position (*x* and *y*, stored as in a PVector)
* a size (*width* and *height*, stored in a PVector)
* a colour

An Obstacle has a constructor that sets the properties appropriately.

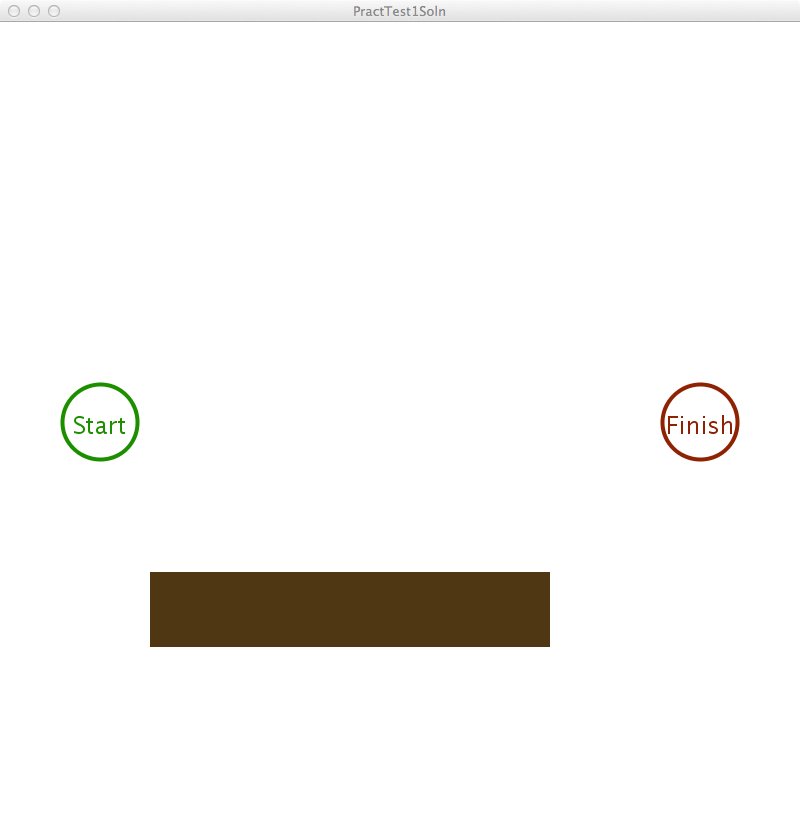
An Obstacle has the following methods:

* A draw() method that draws the obstacle
* A pointInObstacle() method that takes a point (*x,y*) and returns true if that point lies inside the rectangle of the obstacle.

After you have created the Obstacle class, modify the main program in the following ways:

* one Obstacle object should be created in setup() and drawn in the draw() method
* the game should check for collisions between new points added to the trail and the obstacle – if the trail collides with the obstacle, than the trail should be deleted and a game over message printed to the console

The following figure illustrates the game with a single obstacle:



***Save your work as PracTest1\_PartA****.* **Part B [10 Marks]**

***Copy your answer to Part A and save it as PracTest1\_PartB.***

Create a new class called Maze. A Maze object will be an object composed of many obstacles.

The Maze class should have a single property, specifically a list of Obstacle objects from Part A.

The Maze class has a constructor that initializes the list of obstacles.

The Maze class requires three methods:

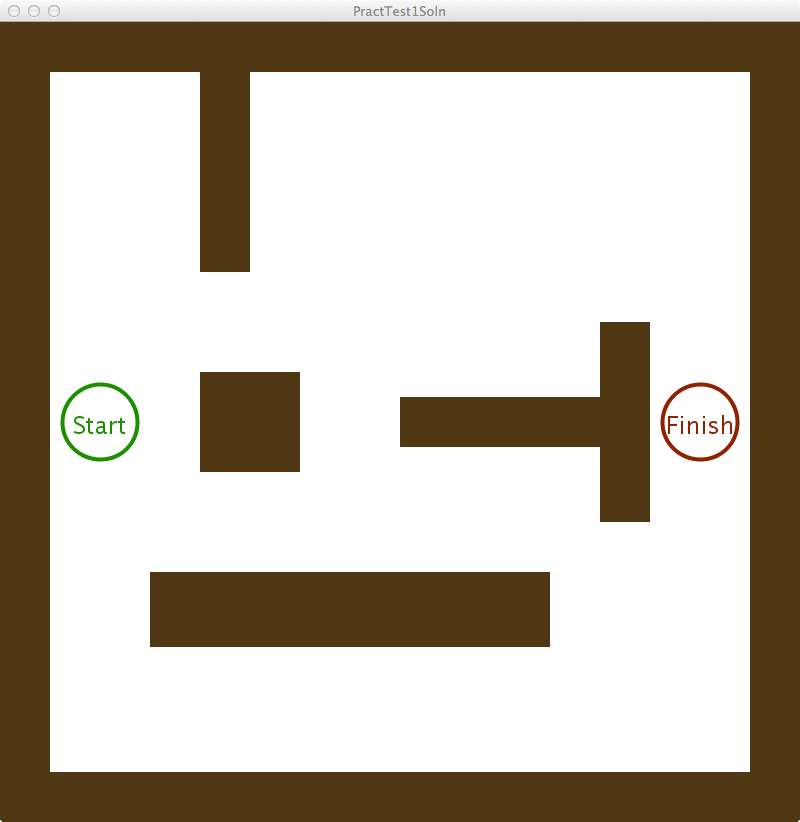
* a draw() method that draws all the obstacles in the list
* an addObstacle() method that has four parameters (*x,y,w,h*) and creates and adds a new obstacle to the list
* a checkForObstacleCollision() method that has two parameters (*x,y*) checks all the obstacles in the list for a collision, returning true is (*x,y*) lies inside any obstacle, false otherwise.

After you have created the Maze class, modify the main program in the following ways:

* delete all references to the Obstacle object that you added in Part A; you will be adding a Maze object instead
* one Maze object should be created in setup() and drawn in the draw() method
* ensure that the maze object has at least six obstacles added to it
* the game should check for collisions between new points added to the trail and the maze – if the trail collides with the maze, than the trail should be deleted and a game over message printed to the console

In other words, the aim of the game is for the user to navigate a trail through the maze from start to finish.

A depiction of what the screen should look like is given here:



Note that your obstacles may be placed in different locations.

***Save your work as PracTest1\_PartB****.*

## Marking Scheme

|  |  |  |
| --- | --- | --- |
| **Part** | **Item** | **Marks** |
| **A** | Obstacle has position property | /1 |
|  | Obstacle has size property | /1 |
|  | Obstacle has colour property | /1 |
|  | Obstacle has proper constructor | /1 |
|  | Obstacle has proper draw() method | /1 |
|  | Obstacle has proper pointInObstacle() method | /1 |
|  | Game works properly | /4 |
| **B** | Maze has a list of Obstacle | /1 |
|  | Maze’s constructor initializes the list properly | /1 |
|  | Maze has a proper draw() method | /1 |
|  | Maze has a proper addObstacle() method | /1 |
|  | Maze has a proper checkForObstacleCollision() method | /1 |
|  | Game works properly (only single Maze object in main) | /5 |
| **TOTAL** |  | **/20** |