

Assignment 3

DUE DATE: MARCH 26TH 2023 AT 6PM

Read through the ENTIRE description for a question (and/or the assignment), take notes on the important points, and plan your code on paper BEFORE writing any code in Processing.

You'll be creating a game where an item is dropped from a travelling object in the sky, and you are trying to catch the item on a target. Points are earned when the item hits the target. This could be a package of food dropped from a plane, a snowflake dropped from a cloud, a present dropped from Santa Claus... use your creativity!

NOTES:

- Name your sketch using your name, the assignment number and the question number, *exactly* as in this example: `LastnameFirstnameA3Q2`. You will **submit only one program in this assignment, for the highest question completed**.
- There are marks for the behaviour of your program and marks for coding. If your program does not run upon download, you will not receive any marks for program behaviour.
- Submit **one PDE file** only, containing the answer to the highest question you completed, which will contain all of the code for the earlier questions. Do not submit any other types of files.
- Assignments must follow the programming standards document published on the course website on UMLearn.
- After the due date and time, no assignment will be accepted.
- You may submit a question multiple times, but only the most recent version will be marked.
- The marker will run your program and may change the canvas size and the constants specified in the assignment. It should still work if any constants are changed in a reasonable way.
- These assignments are your chance to learn the material for the exams. *Code your assignments independently*. We use software to compare all submitted assignments to each other and pursue academic dishonesty vigorously.

Each question builds on the previous. Make sure that you follow ALL instructions carefully and finish one question completely before moving on to the next question. You are asked to do certain things in a specific way so that you don't run into difficulty later.

The purpose of this assignment is to practice using functions with parameters and return types. **You must pass data to and from functions as described, and limit the use of global variables to data that must be remembered from one frame to the next.** (Global final constants are also OK.)

Remember that one of the main benefits of functions is that you code them one at a time, without worrying about how they will be used. When you are asked to write functions, write each function and then decide how the functions should interact to make the game behave as described.

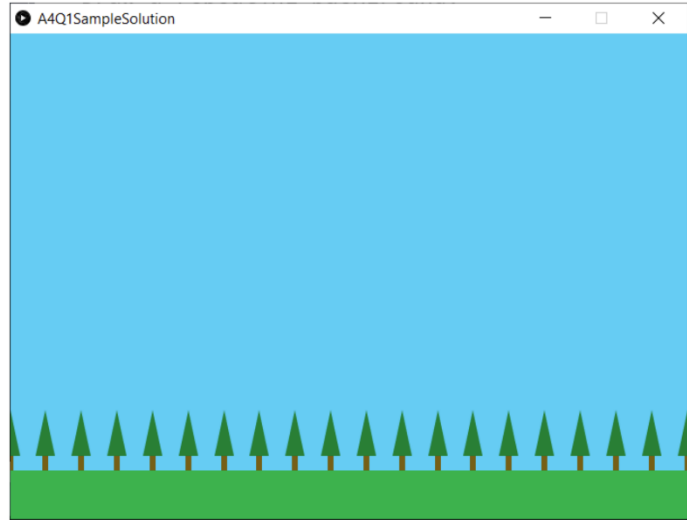
Your first task in each question is to design the program. Read through the requirements carefully. Make a list of functions you will use, and plan out how the functions will work together to obtain the desired outcome. Write the code one function at a time, testing each thoroughly as you go.

There are some hints on how to design the program in the instructions below, but there are many ways to solve a problem, and there is not a single correct approach. Follow good programming practices. In particular, each function should perform only a single task, and each variable's scope should be as small as possible.

Q1: REPEATING BACKGROUND [3 MARKS]

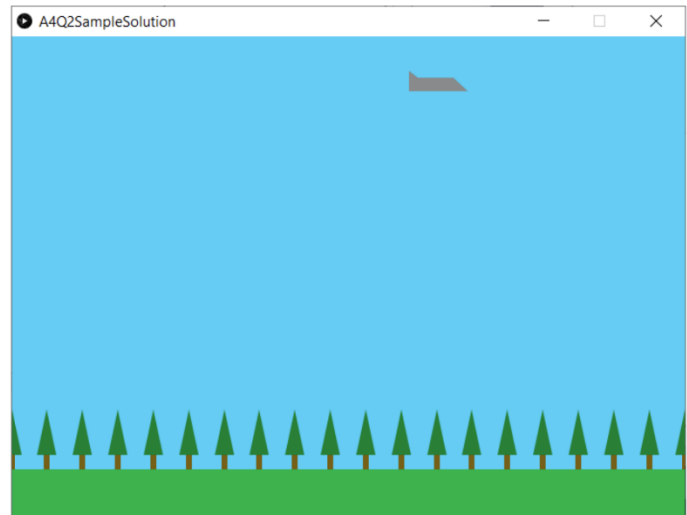
Create an Active Processing program that will draw some kind of repeating background. *Choose your own background, be creative!*

- Create the usual `setup()` and `draw()` functions.
- From `draw()`, call a function `drawBackground()` that controls the drawing of the game background.
- `drawBackground()` should
 - Draw a horizon (e.g., the grass in the example).
 - Draw a repeating set of items (e.g., the trees in the example). This should be done by
 - Writing a function that accepts the location of one item (as parameters), and draws the item at the given location.
 - Repeatedly calling the function to draw one item, each time passing a different position, to draw all of the items in the background.
- The repeating item (e.g., tree in the example) must consist of at least two shapes.
- All variables used in this question must be local. You should not use any global variables in Q1.
- All dimensions should be relative to the canvas size, so that if the canvas size is changed, the size of all the items drawn will also change.

**Q2: FLYING OBJECT [3 MARKS]**

Add an object that flies from left to right, over and over.

- The object can be simple but should be of your own design. It should use at least three shapes (e.g., the plane in the example uses one rectangle and two triangles).
- The object's coordinates should be stored globally, because we will need to remember where the object is from one frame to the next.
- A `drawXXXXXX()` function (where `XXXXXX` is your object) should use the current position of the object to draw the object. The object should be near the top of the canvas.
- A `moveXXXXXX()` function (where `XXXXXX` is your object) should update the position of the object, using a `SPEED` constant to determine how much the object should move from one frame to the next.
- The `moveXXXXXX()` function should make the object wrap, so that once it has gone off the right side of the canvas, it reappears from the left side of the canvas. The object should move completely out of view (off the canvas) before reappearing, and when it reappears, it should appear to emerge from the left side (i.e., you should not see the entire object pop into view: it will gradually fly off the right side of the canvas, then fly onto the canvas from the left side).
- `moveXXXXXX()` and `drawXXXXXX()` should be called from `draw()`.



Q3: DROP AN ITEM [7 MARKS]

In many programs in this course, an item like a moving ball has been controlled by keeping track of its position (x and y), and making a small change to that position each frame. That will NOT be done for the item dropped in this assignment. Instead, formulae can calculate the position of the item at any instant in time, using its initial velocity v (in pixels per second), the gravitational constant g (in pixels per second-squared), the initial x and y positions x_0 and y_0 (the position of the flying object at the time the item is dropped) and the time t (the number of “seconds” since the item was dropped) using the following formulae:

- The initial velocity in the x direction is v_x .
- The initial velocity in the y direction is 0 (the plane is moving horizontally)
- The x position at time t is $x_0 + v_x t$
- The y position at time t is $y_0 + 0.5 g t^2$ (note that this is the position measured in Processing coordinates – y increases as the item falls)



In this game, time will be measured as the number of frames elapsed since the program began (and because there are many frames per second, the game “seconds” do not correspond to clock time). Use a global variable to keep track of the time (we need to remember it from one frame to the next). Because the game “seconds” are not real seconds, and the unit of measurement is pixels rather than a real distance, gravity is also different from the constant you might be familiar with from physics. Try `GRAVITY = 0.02` and adjust as desired.

Drop the item when the player presses the “Enter” key. At that moment, store the initial position of the item, and the time the item was dropped. While the item is in motion, in each frame, calculate the position of the item using the functions below. Use the coordinates returned from those functions to draw the item at the correct position.

The following functions should use the above formulae to calculate the current position of the item:

- `float calcTimeSinceDropped(float time)` should calculate the time since the item was dropped. The time passed to the function should be the time the player hit Enter.
- `float calcItemX(float flightTime, float initX, float initV)` (where `Item` is your item) should calculate the x coordinate of the item at the given `flightTime` (t in above formulae), with the given initial x position (x_0) and initial velocity (v_x).
- `float calcItemY(float flightTime, float initY)` (where `Item` is your item) should calculate the y coordinate of the item at the given `flightTime` (t in above formulae) with the given initial y position (y_0).

You must use the above functions in your program and calculate the coordinates of the item in every frame. DO NOT store the item’s current coordinates globally.

When an item is in the air, print the coordinates in the top left of the canvas, as in the example image. In this question, if the item moves off the canvas, it should continue to move out of sight, with the coordinates telling the player where it is.

Use a boolean variable to make sure that while one item is in motion, another item cannot be dropped (i.e., nothing will happen if the user presses Enter).

Q4: ADD A TARGET [9 MARKS]

Draw a round target at the mouse's x position within the horizon area (the green area in the example). The target should follow the mouse's x position during gameplay.

Write a function `boolean itemInObject(float objectX, float objectY, float objectDiameter, float itemX, float itemY)` (where `item` is your item) that will test if the item's position is within a circular object (specified by its center coordinates and its diameter).

Each frame when the item is in flight, use the `itemInObject` function to test if the item has hit the target. If the item has hit the target, increase the score (which should be displayed in the top right), print a message (see below) and the boolean variable that tracks whether there is an item in motion should be set to false.

Write a function `boolean belowGround(float y)` that will test if the item's y coordinate is below ground level (off the bottom of the canvas), indicating that the item has missed the target.

Write a function `boolean outOfView(float x)` that will test if the item's X coordinate is off the right side of the canvas. Because the target must be on the canvas, this also indicates that the item has missed the target.

Each frame when the item is falling, use the `belowGround` and `outOfView` functions to test if the item has missed the target.

At key points in the game, a message should be printed on the canvas, as show in the example. The message should be displayed for only a brief time after an event (long enough for it to be read, at least 60 frames) and then disappear.

Possible messages should be:

- "ITEM RELEASED" (where ITEM is your item) when an item is dropped from your flying object.
- "TARGET HIT" when the item hits the target.
- "MISS!" when the item hits the ground without hitting the target, or goes off the right edge of the canvas.

Q5: ADD LEVELS [2 MARKS]

After every three points, increase the level in the game. The target should become smaller (until a minimum size is reached after approximately ten levels, at which point the target size will remain the same to the end of the game).

PROGRAMMING STANDARDS [6 MARKS]

Assignments must follow the programming standards document published on the course website on UMLearn, as well as any standards described in the lectures (e.g., `CONSTANT_NAMES` vs `variableNames`). Make appropriate use of functions, where each one does one thing, and one thing only.



HAND IN

Hand in **one pde file only**, containing the answer to the highest question you completed, which will contain all the code for the earlier questions. It must run without errors, or you will lose all of the marks for the test run. Please clearly indicate in your comment block at the start of the program the highest question number you have completed.

Make sure your file is correctly named, **exactly** as specified in the instructions at the beginning of the assignment. You may be penalized for an incorrectly named file. Make sure there are no extra characters, such as spaces and underscores.

The marker will run your program and may change the canvas size and the values of the constants. It should still work if any constants are changed in a reasonable way. You may be significantly penalized if your program doesn't run, and substantially penalized if it does not work with different values.