Mini-Assignment 2: Processing

Handed out: Wednesday, 8 October 2014
Due: Friday, 17 October 2014

In this assignment, you will use Processing to recreate the classic "tank wars" video game in which opposing players launch projectiles to hit and destroy their opponent's tank. You are given a skeleton of the game, declaring certain variables and functions. In the following steps, you will fill in the missing code to make a fully functioning game.

How the game is played:

The rules are simple. Players take turns launching projectiles at each other. The first player to strike the other player's tank wins. Each player controls the angle and strength of the projectile launch; the normal rules of gravity and object collisions determine where it goes.

Instructions:

First, make sure you have the "tank_wars" Processing sketch. When you run it initially, you should see a blank window with some text at the top. Take some time to familiarise yourself with the code. The variables at the top of the file hold the global *state* describing what's happening in the game. The behaviour of the `setup()` and `draw()` functions should be familiar from class. Notice that `draw()` calls several other functions in turn to draw each aspect of the game. You will be responsible for implementing many of these functions.

Step 1: Draw the terrain and the tanks
In this section, you will implement the `drawGround()` and `drawTanks()` functions. The height of the ground has been calculated randomly at the beginning of the game and stored in the `groundLevel[]` array. The array is indexed by horizontal position: for example, `groundLevel[3]` will tell you the level (Y) of the ground at X = 3 (the fourth pixel from left-- remember the pixels start numbering from 0).

Implement `drawGround()` so that it colours the terrain a dark grey. *Hint: think about using the line() function to draw a vertical line at each X location.*

Information on the tanks can be found in the variables at the top of the file. Draw Player 1's and Player 2's tanks at the positions stored in the relevant variables, with the size specified in `tankDiameter`. Draw each tank as a semicircle (*Hint: arc()*), using different colours for each.

You should also draw the cannons for each tank. The cannon angles are stored in `tank1CannonAngle` and `tank2CannonAngle`. Draw the cannon at the correct angle using `line()` and the `sin()` and `cos()` functions to calculate the right coordinates.[1] For the default angles, the cannons should point straight up. Your may need to adjust the signs to achieve this behaviour. *Hint: strokeWeight() can be used to draw a thicker line.*

---

[1] See here for a quick intro to using trigonometry (*sin* and *cos*) to calculate coordinates: http://www.mathsisfun.com/algebra/trigonometry.html

Step 2: Handling events at the keyboard
Every time a key is pressed, the `keyPressed()` function will be called. In this step, you will fill in the contents of this function to let the players aim and fire their tanks. The left and right arrow keys should adjust the angle of the cannon; the up and down arrow keys should adjust the strength of the eventual launch, and the space bar should launch the projectile.

You can use the `key` variable to determine which key was pressed. The arrow keys will return a special value of `CODED`; in this case, the `keyCode` variable can be used to determine which arrow was pressed. See the Processing reference (and the built-in examples) for further details.

Which tank is adjusted depends on whose turn it is. That information is stored in the `player1Turn` variable, which is a boolean where `true` indicates Player 1 is currently up.

For now, when the space bar is pressed, don't fire the projectile: just have it switch to the next player's turn. The function `nextPlayersTurn()` has been provided to do this.

Step 3: Fire a projectile
Revisit the `keyPressed()` function. This time, when the space bar is pressed, a projectile should fire. Remove `nextPlayersTurn()` from `keyPressed()`. You will also need to do three things:

1. Set the initial position and velocity of the projectile by setting the value of the appropriate variables at the top of the file. The velocity should depend on the angle and strength of the current player's cannon. (*Hint: use sin() and cos() again.*)
2. Set the boolean variable `projectileInMotion` to `true` so your code knows later that it should draw the projectile and update its position
3. Add code to `updateProjectilePositionAndCheckCollision()` so that each time it updates the position of the projectile according to its velocity.

Note that the code to draw the projectile has been given to you (though you are welcome to change its appearance if you like).

Step 4: Gravity
In the previous step, when you hit the space bar, you should see a projectile continue in a straight line until it leaves the screen. In this step, add gravity so the projectile returns to the ground. This can be implemented in one or two lines in `updateProjectilePositionAndCheckCollision()` so that the velocity updates on each call. A `gravity` variable has been provided at the top of the file.

Step 5: Collision Detection
In the previous step, the projectile should have returned to the ground and continued off the bottom of the screen. In this step, you'll check for two collisions: with the ground and with a tank.

You can assume for practical purposes that the projectile is a point source (i.e. you can consider only the location of the projectile and not its diameter). However, for the tanks you need to consider their diameter. (*Hint: how would you detect if the projectile is within the area of a circle or semicircle?*)

For the ground, you can use the `groundLevel[]` variable to detect when the projectile has passed below the surface of the terrain (how?). *Hint: the array index needs to be an integer value. You can cast a floating point value to an integer by preceding it with* `(int)`. *See setup() for an example.*

Important: when a collision is detected, the projectile should stop moving. You can control that with the `projectileInMotion` variable. When the projectile hits the ground, you should advance to the next player's turn. When the projectile hits one of the tanks, the opposite player wins. (Yes, it's possible to destroy your own tank!) Use the `playerHasWon` variable to set who wins.

Congratulations: you should now have a fully functioning game!

Step 6: Customise your game
As a final step, choose one (bonus: two or more) of the following ideas to customise the game play. Ideas are listed in rough order from easiest to hardest.

- Make the appearance of the tanks, projectile or terrain more interesting.
- Add an effect of wind on the horizontal projectile velocity. Randomise the wind on each turn and display it at the top.
- Change the interaction so the game is played with the mouse rather than the keyboard (use the functions `mouseMoved()` and `mouseClicked()`).
- Make the projectile explode (grow in diameter) when it collides with something. If the explosion reaches a tank, it should be considered a hit.
- Make a more varied terrain pattern with hills and valleys.
- Change the game to let both players play simultaneously, using different keys. Two projectiles could thus be in the air at once (and even collide with each other).
- Make the projectile blow a hole in the terrain when it strikes.
- Anything else you can come up with!

Your submission should consist of the following:

- Your Processing sketch (only the final sketch is needed for this assignment)
- **Comments** in your Processing code!
- Short report (PDF, one side of A4) describing your procedure and findings: challenges encountered, notable design decisions. The structure and content of this is relatively open (your code will be the most important deliverable), but try to provide a sense of how you went about the project, especially in steps you found challenging.