

# Stanford University, CS 193A

## Homework 6: Snake

### (Pairs allowed)

*Assignment spec and idea by Marty Stepp.*

This assignment practices 2D graphics and games. Make sure to check out the **Development Strategy** at the end of the document for tips on getting started. We suggest using our [Stanford Android Library](#) and its GCanvas, GSprite, etc. for this assignment, though you are not required to do so if you prefer another library or doing your drawing code from scratch.

This is a **pair assignment**. You may complete it alone or with a partner. Please read our [Pairs web page](#) for some of our policies and suggestions if you do plan to work in a pair.

### Starter Files:

-  [snake-starter-files.zip](#) (ZIP of all starter files)

### File and Project Names:


- Your **Android Studio** project's name must be EXACTLY the following, case-sensitive, including the underscores:

- cs193a\_hw6\_**SUNETID** (example: cs193a\_hw6\_ksmith12)

Or if you are working in a pair, please list both partners' SUNetIDs separated by an underscore:

- cs193a\_hw6\_**SUNETID1\_SUNETID2** (example: cs193a\_hw6\_ksmith12\_tjones5)

- You must have activity/view classes with the exact names given below under "Program Description," case sensitive, in corresponding file names (e.g. class FooActivity in file **FooActivity.kt**). You may add other activities, views, classes, and files to your app if you like, but you must have the exact activity file names below at a minimum, spelled and capitalized exactly as written here.
- When you are finished, ZIP your project ([instructions](#)) into a file with the following exact file name, including spelling and capitalization:
  - cs193a\_hw6\_**SUNETID**.zip (example: cs193a\_hw6\_ksmith12.zip)

- If you use **libraries** from the Maven repository (ones that are added to a project by declaring them in your **build.gradle** file), these will be auto-downloaded when we compile your project to grade it. If you use any other local libraries that you downloaded manually, make sure they are located in **app/libs/** and are included in your ZIP that you turn in.
- Turn in link:  [Turn in HW6 here.](#)

## Program Description:

For this assignment you will write a game app named **Snake**. Snake is a classic 2D mobile phone game where you control a snake that moves around on the screen eating food. The snake has a given constant velocity in one of the four directions (up, down, left, or right) and can be "turned" using two buttons, each of which changes the snake's direction by 90 degrees. At any given time during the game, there is a piece of **food** available at some randomly chosen location on the screen. If the snake moves to that location and touches the food, he eats it. The player scores a point for each piece of food eaten. Eating a piece of food also causes the snake's **tail** to **grow longer**. If the snake bumps the edge of the screen or if his head bumps into some part of his ever-growing tail, he **dies** and the game is over.

We would like you to write at least these two activity classes:

- **TitleScreenActivity**: Shows the app name, logo, and current high score, and has buttons for the user to start playing or exit the app. (To exit an app/activity, just call its `finish()` method.) When the user presses the Play button, it launches the **SnakeGameActivity**.
- **SnakeGameActivity**: Lets the user actually play the game. Most of the screen is taken up by a **SnakeCanvas**, which is a graphical canvas or custom view that draws and animates the gameplay. Under the **SnakeCanvas** are two buttons that allow the user to turn the snake left and right to control its movement.



TitleScreenActivity

SnakeGameActivity

*Screenshots of app's various activities*

You do not need to exactly match the appearance of the screenshots in this document. Please feel free to be creative and customize your app as you like, so long as your app contains the functionality described below.

## Implementation Details:

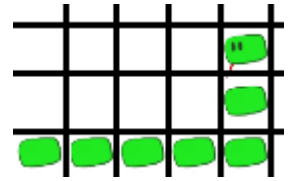
*Drawing the sprites:* In our starter ZIP we provide some images you can use as bitmaps in your code. You are not required to use these, but if you want them, place the images in **res/drawable/**, such as the following:

- **snakelogo.png** - game logo to show in title screen
- **food.png** - picture of the food (an apple)
- **foodcheese.png, foodpizza.png, ...** - pictures of other kinds of food
- **rotateleft.png, rotateright.png** - icons that can go on the buttons for rotating the snake
- **snakehead.png, snakebody.png** - images to represent the snake's head and his body segments

You may find that a particular image is not the size you want it to be. If so, use the [\*\*Bitmap.createScaledBitmap\*\*](#) method to resize it to a more suitable size.

*Grid and movement:* We suggest that your code thinks of the game's drawing canvas as though it were a grid of squares of some size, such that game sprites position and move themselves by

entire "squares" worth of pixels at a time. On each frame of animation, the snake moves by exactly 1 frame in his current direction. This makes the coding, movement, and collision detection easier. The screenshot at right shows the idea of the grid (though the grid itself is not supposed to be drawn on the screen).



There are several ways to implement the snake's movement. We suggest making the head of the snake into a [GSprite](#). Each time the game's animation ticks, the head will move itself equal to the size of one square in the grid. In our lecture code for games and animation, we used **velocity** to do sprite movement. But for the snake segments, velocity might be harder to use because the pieces move in a coordinated way: each piece needs to move forward to where the piece in front of it used to be. You can loop over the snake tail segments if you keep them in a list, and each segment can be told to move into the location of the next segment in line. If you do use velocity to help you with movement, you may want to also use the [GSprite](#) method `rotateVelocity` to handle left and right turns.

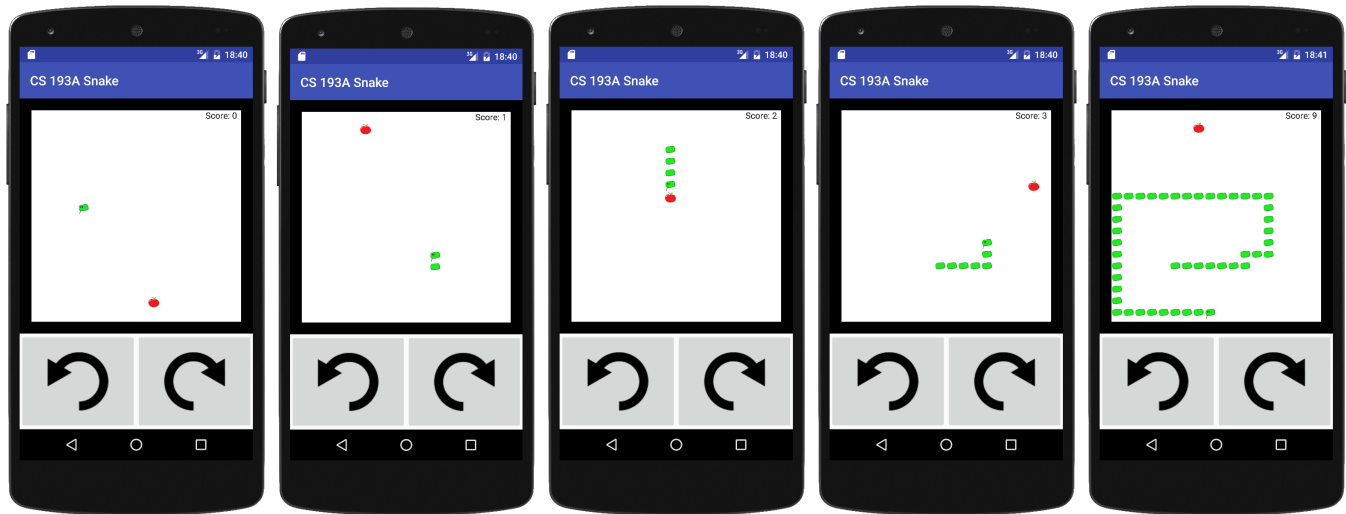
*Animation:* You can do animation using the `GCanvas`'s `animate` method or a thread. Since movement is in larger chunks in this game (a grid square at a time), you should probably use fewer frames per second than the animations shown in class. A value of 2-3 FPS is good for testing, and perhaps 4-6 FPS for a finished game to make it more challenging. (You might consider slightly increasing the FPS as the game goes on to make it progressively harder for the player.)

*Food and eating:* There should always be 1 piece of food at a random square on the game world. You can position a sprite to a random location with a `Random` or [RandomGenerator](#) object. Move it to a location that is aligned on the boundaries of your grid squares. You can do this by computing how many grid squares there are in each dimension, picking a random row and column in that range, and then multiplying it by the size of a grid square to get the x/y pixel coordinates. Make sure not to move the food to a location that is already occupied by part of the snake!

If you're using our `GCanvas` and `GSprite` libraries, you can check whether the snake's head collides with the food using the `collidesWith` method. If not, consider using Android's `RectF` class and its `intersects` method. To make the snake die when he hits the edge of the game world, you could use the `isInBounds` method of a `GSprite`, or you could put some dark rectangular `GSprites` around the edge of the screen to represent walls and then check whether the snake's sprite collides with them.

If you are getting unwanted collisions, such as when the snake is near (but shouldn't be touching) the food or screen edge, consider using the `GSprite`'s collision-margin feature to give it a smaller collision rectangle than the sprite's onscreen appearance. You might also consider shrinking your sprite size to be 1-2 pixels smaller than your grid size, to make sure that a sprite never accidentally bleeds into an adjacent grid square.

*Growing tail:* Eating a piece of food also causes the snake's **tail to grow longer**. The following series of screenshots shows the snake's tail growing longer as he eats more and more food:



*Screenshots of snake's tail growing after eating food*

If you want a simpler game, grow the snake's tail by 1 segment for each piece of food eaten. A more sophisticated game grows the tail by  $N$  segments when the snake eats his  $N$ th piece of food. For example, eating the first food grows the tail by 1 segment, eating the 2nd piece of food grows the tail by 2 segments, and so on.

*Game Over and High score:* Display the score on the screen during gameplay; if you are using GCanvas, do this with a [GLabel](#). This same label can be used to show other game messages, such as "Game over" when the snake dies.

When the game ends, the snake should stop moving and some kind of "Game over" message should be displayed. You do not need to do anything else when the game ends, such as letting the user start again, but you can if you like. Your game can remember the highest score ever earned by the player. Display this on the title screen when the game loads, and update it if the player ever beats this highest score. You can save the high score using Shared Preferences as shown in lecture, or by writing it to a text file in the device's storage.

## Generalizing the App for Different Devices:

For full credit, your app must function properly under all of the following kinds of conditions:

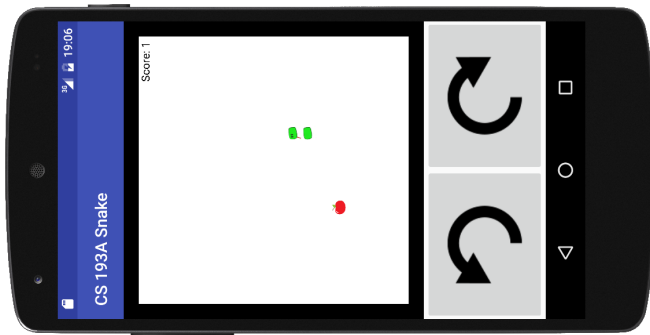
*Change of orientation:* By default you'll find that your game doesn't work very well in landscape mode. Since the screen's width/height ratio is so different in landscape mode, the game's canvas will be too small, or the left/right turn buttons will be obscured, or both. You should do something to make this experience better for the user.

The simplest but non-ideal thing you can do is to set the app not to rotate certain activities, forcing the phone to stay in portrait mode even if the device is rotated. You can do this by

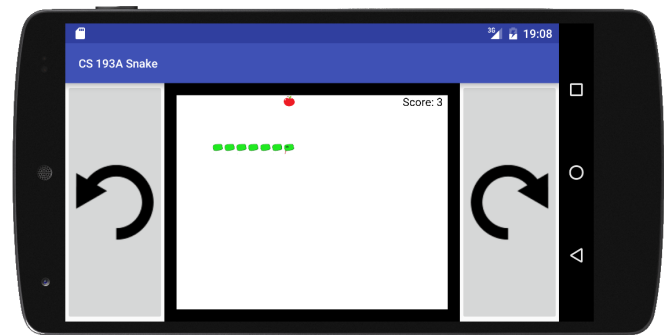
editing your **AndroidManifest.xml** file and setting the activity's screen orientation attribute to portrait as shown below:

```
1 <activity android:name=".ActivityClassName"
2         android:screenOrientation="portrait" />
```

A better approach would be to create a **new layout** for landscape mode that uses the screen space better, placed in the **layout-land** resource folder. The second screenshot below shows one example layout where the Left/Right turn buttons have moved to the left and right edges of the screen, rather than being underneath the game's canvas. This gives each element more space and would be more ergonomic for playing the game if the user were holding a real phone. Note that you do not need to use **fragments** like you did in Friendr; a fragment was needed there because we used a different activity altogether rather than just a different layout for an existing activity.



*with activity screen orientation set to portrait*



*with special landscape layout*

*Different device sizes:* Make your code general enough that it will work well on devices of various screen sizes. You don't need to support every possible device, but a good pair to support would be a standard Nexus 5 style phone and a larger-screen tablet device like we discussed in class. Make the game graphics **scale** themselves appropriately so that the game doesn't look too large or too small on each device.

## Extra (Optional) Features:

It's already a lot of work to build the basic game and add the previously described features like scaling. But if you want to extend this assignment further, here are some ideas, or come up with your own:

- **Sound/Music:** In our starter ZIP we provide a few sound effect files that you can put into the **res/raw** folder of your project.
- **Rotated images:** Make it so that when the snake turns up, down, left, and right, his head image turns to face that way. We haven't learned how to rotate a **Bitmap** in class, but you can learn how on Google.



- **Smoother animation:** Make the snake move smoothly at a high frame rate such as 60 fps, rather than being locked down to a grid as we suggested earlier. This can make the coding harder, especially for moving the snake's body pieces, but it can make the game look really nice if you do it right.
- **Keyboard controls:** Make it possible to use arrow keys to steer the snake.
- **Localization:** Localize the text and some images in your game so that it looks good in at least one other locale. We have provided some **extra images** in our starter ZIP that could be used for another locale. To use these for localization you'd need to rename and move them. For example, to do a French version of the game, make a **res/drawable-fr** folder and put the **snakelogo\_fr.png** in that folder with the name of **snakelogo.png** and the **foodcheese.png** in there with the name of **food.png**. The file names have to be the same as the ones in the main **res/drawable** folder for Android to use them properly. Or you can download your own images if you prefer.

Also localize the **game text** using resource strings for your language. You aren't being graded on your language abilities, so if you don't remember the translations for the strings, either look them up on a service like [Google Translate](https://www.google.com/translate) or just type in any silly message you want.

If you do add localization to your game, make sure that your game's **title screen** tells the user what languages are supported. Most apps wouldn't bother to do this, but in our case, it will help the grader running your program quickly figure out what they should set their device/emulator's language to if they want to see the localized features.

- **Additional gameplay features:** A snake that fires bullets? Multiple pieces of food on the screen? A counter of lives that you can lose before getting a Game Over? Get creative and enhance the game.

## Development Strategy:

Here are some suggested steps and milestones for getting started on the assignment:

- *Snake head and activities/views:* Make the snake's head appear on the screen at a reasonable location and size. Check that the snake game activity loads up and shows your view properly.
- *Movement:* Make the snake able to move forward and set up a basic animation loop. Make the snake change his direction using the Turn L/R buttons.
- *Food:* Make a piece of food appear randomly on screen. Log a message when the snake touches the food.
- *Eating:* Make it so that the act of eating (colliding with) the food causes it to respawn randomly and gives the player a point.

- *Lengthening the snake:* This one can be tricky. Make the snake get longer when he eats. One way you could do this is by keeping a boolean flag such that when the snake hits the food, on his next move a new tail piece will appear where the end of the snake used to be.
- *Game flow:* Make it possible for the snake to die and end the game by hitting his own tail or walls.
- *Other features:* Work on scaling, orientation, etc. You can do it!

## Style Requirements:

Here are some important style requirements below that we ask you to follow. If you do not follow all of these requirements, you will not receive full credit for the assignment.

- **Comments:** Write a **comment header** in your main activity **.kt** file containing your name and email address along with the name of your app and a very brief description of your program, along with any special instructions that the user might need to know in order to use it properly (if there are any). Also write a brief comment header at the top of every **method** in your **.kt** code that explains the method's purpose. All of these comments can be brief (1-2 lines or sentences); just explain what the method does and/or when/why it is called. You do not need to use any particular comment format, nor do you need to document exactly what each parameter or return value does. The following is a reasonable example of a comment header at the top of a **.kt** activity file:

```
1 /*
2  * Kelly Smith <ksmith12@stanford.edu>
3  * CS 193A, Winter 2049 (instructor: Mrs. Krabappel)
4  * Homework Assignment 6
5  * NumberGame 2.05 - This app shows two numbers on the screen and asks
6  * the user to pick the larger number. Perfect for Berkeley students!
7  * Note: Runs best on big Android tablets because of 1000dp font choice.
8  */
```

- **Redundancy:** If you perform a significant operation that is exactly or almost exactly the same in multiple places in your code, avoid this redundancy, such as by moving that operation into a helping method. See the lecture code from our lecture #1 for examples of this.
- **Reduce unnecessary use of global variables and private fields:** While you are allowed to have private fields ("instance variables") in your program, you should try to minimize fields unless the value of that field is used in several places in the code and is important throughout the lifetime of the app. Our lecture code from Week 1 shows examples of reducing fields by accessing values from widgets instead (using `getText` and similar). (We consider it okay to declare widgets as fields if you use the ButterKnife library and its `@BindView` annotation.)



- **Reduce mutability:** Use `val` rather than `var` as much as possible when declaring Kotlin variables. Use immutable collections instead of mutable ones where appropriate.
  - **Naming:** Give variables, methods, classes, etc. descriptive names. For example, other than a for loop counter variable such as `int i`, do not give a variable a one-letter name. Similarly, if you give an id property value to a widget such as a `TextView`, apply a similar style standard as if it were a variable name and choose a descriptive name.
- 

*Survey:* After you turn in the assignment, we would love for you to fill out our optional [anonymous CS 193A homework survey](#) to tell us how much you liked / disliked the assignment, how challenging you found it, how long it took you, etc. This information helps us improve future assignments.

*Honor Code Reminder:* Please remember to follow the **Honor Code** when working on this assignment. Submit your own work and do not look at others' solutions. Also please do not give out your solution and do not place a solution to this assignment on a public web site or forum. If you need help, please seek out our available resources to help you.

*Copyright © Stanford University and Marty Stepp. Licensed under Creative Commons Attribution 2.5 License. All rights reserved.*