Programming Assignment 8 (100 Points)

11:59pm Thursday, November 17th
START EARLY!

Overview:

This programming assignment is a simplified version of the classic game Snake. For those that are not familiar with the game Snake, it is a game where the snake grows size as it eats more food. The goal is to make the snake so big that it covers the entire grid. When you start the game, the snake will be continuously moving in the direction you tell it to. The snake will grow if it eats food, which will be randomly placed on the canvas. If the head of the snake crashes into a part of its body (it tries to eat itself), then the game ends. In this specific version of the game, let's say that if the snake hits a border, it also crashes and the game should end.

For a more general idea of how Snake works, follow this Wikipedia Link. Remember, the Wikipedia link **is NOT** a substitute for the write-up; it should only be used to understand how the game in general should work. Implementation of this game **MUST** follow the exact specifications found in this write up.

README (10 points)

You are required to provide a text file named **README,** NOT Readme.txt, README.pdf, or README.docx, etc. with your assignment in your pa8 directory. There should be no file extension after the file name "**README**". Your README should include the following sections:

Program Description (3 points) :
Explain how the user can run and interact with each program. What sort of inputs or events does it take and what are the expected outputs or results? How did you test your program? How well do you think your program was tested? Write your README as if it was intended for a 5 year old or your grandmother. Do not assume your reader is a computer science major.

Short Response (7 points) : Answer the following questions:

Academic Integrity Related Questions:

1. How do you maintain your integrity even when you are stressed, pressured, or tired?

Java Questions:

2. A high school student is trying to write a Java program that will draw different shapes and in different possible colors. To do this, she has written just one Java class called ShapeDrawer, which contains all the necessary methods like drawRedCircle(), drawBlueCircle(), drawYellowSquare(), drawGreenSquare(), and so on. Using object-oriented terminology, describe how you can help the student improve her design.
3. List at least two ways in which Java Interfaces and Java abstract classes are **different**.
4. List at least two ways in which Java Interfaces and Java abstract classes are **alike**.

Vim Questions:

5. How can you run gvim through the command line to open all Java source code files in the current directory, each file in its own tab?

Unix Questions:

6. Suppose you are currently inside a directory and in there you want to make a new directory called fooDir. And inside fooDir, you want another directory called barDir. Using only a **single** mkdir command, how can you create a directory called fooDir with a directory called barDir inside it?

STYLE (20 points)

Refer to the PA2 writeup for a detailed style guide.

CORRECTNESS (70 points)

All of your code/files for this assignment need to be in a directory named **pa8** in your cs11f home directory. Please see previous programming assignments to setup your **pa8** directory, compile, and run your programs.

0 – Get the Starter Code:

Copy the files from `~/../public/PA8_Starter_Code/` into your `~/pa8/` directory.

START EARLY!

This will be a long assignment!

1 – Command-line Arguments:

Before setting up the GUI, we first begin with implementing the command line arguments.

For this assignment, instead of hard-coding the dimensions of the canvas and such into the Controller class, we will pass them in as command line arguments.

What are command line arguments? Check out the oracle tutorial page.

- There shall be 3 arguments: the dimensions of the canvas (width and height), and the delay.
  Usage: `java -cp ... SnakeController [width] [height] [delay]`

- Example of valid arguments:
  `java -cp ./Acme.jar:./objectdraw.jar:. SnakeController 300 300 150`

How do you parse the arguments so that they can be passed into the program?

- In java, the command line arguments are stored in the `String [] args` array passed in as a parameter to `main(String[] args)` in the order that they are written on the command line.

- Since we need the values from the command line arguments to initialize the SnakeController, you will need to pass the `args` obtained from `main()` to the constructor of `SnakeController.java`.

  This means that your constructor will look like: `SnakeController( String[] args )`
  The first argument is in `args[0]`, the second is in `args[1]`, and so on.

Error checking for command line arguments

- Before we can start up the program, we need to check for **ALL** errors in the command line arguments. If any of the arguments fail a requirement, print the appropriate error message. After checking all the values, print the usage, then exit the program using:
  ```
  System.exit(1);
  ```
- All the error messages are given to you in `PA8Strings.java`.

- **Number of arguments**: Before we can check the command line arguments, we need to determine how many arguments were passed in. If we don't have the correct number of arguments, print the corresponding error message, usage string, and exit with system status 1 (as shown above).

- **Dimensions:** We can parse the dimensions using the following lines:
  ```
  windowWidth = Integer.parseInt( args[0] );
  windowHeight = Integer.parseInt( args[1] );
  ```

  - Validate that both the width and height variables are between 200 and 800 inclusive
    - Hint: implement a checkRange function, you'll need it for other things too
    - If the variable is not in that range, print the OUT_OF_RANGE String from the PA8Strings.java file provided as an error message.
  - Also check that the dimensions are both multiples of 50.
    - If either variable is not a multiple of 50, print the NOT_EVENLY_DIVISIBLE String from the PA8Strings.java file.
  - Use the following example to format your error messages:
    ```
    System.err.format(PA8Strings.OUT_OF_RANGE, valueName, value, low, high);
    ```

    Note: the arguments after the `OUT_OF_RANGE` string should be what you are passing into the `OUT_OF_RANGE` string, in place of the `%s`, and the `%d`. When you pass these additional arguments in the format call, it replaces the placeholders (`%s` and `%d`) with the value of the passed in variable. For more info, check out this tutorial page and look at the format method section.

- **Delay**: The third argument, delay, represents the amount of time to pause in each iteration of the while loop in the Snake's run() method. The delay must be between 50 and 600 inclusive. Remember that 50 will correspond to a faster snake speed (because the time delay is shorter) and 600 will correspond to a slower snake speed (because the time delay is longer). If the delay is not in the correct range, then print the OUT_OF_RANGE error message.

Note for Testing:

- We **WILL NOT** be testing your command line argument processing with anything besides integers.

- Since we will be providing the strings for you, the print statements MUST look **EXACTLY** as those in the examples. **NO leniency** will be given to those that do not match the examples, and you will **lose FULL points** per test regarding input validation.

SAMPLE OUTPUT : (user input shown in **BOLD**)

1. No arguments

```
[cs11fxyz@ieng6-201]:pa8$ java -cp ./Acme.jar:./objectdraw.jar:. SnakeController
Usage: java -cp ./Acme.jar:./objectdraw.jar:. SnakeController WIDTH HEIGHT DELAY

[cs11fxyz@ieng6-201]:pa8$
```

2. Invalid delay only

```
[cs11fxyz@ieng6-201]:pa8$ java -cp ./objectdraw.jar:./Acme.jar:. SnakeController 600 400
1
Error: delay value 1 is out of range. It should be between 50 and 600
Usage: java -cp ./Acme.jar:./objectdraw.jar:. SnakeController WIDTH HEIGHT DELAY

[cs11fxyz@ieng6-201]:pa8$
```

3. Invalid width only

```
[cs11fxyz@ieng6-201]:pa8$ java -cp ./objectdraw.jar:./Acme.jar:. SnakeController 100 200
50
Error: width value 100 is out of range. It should be between 200 and 800
Usage: java -cp ./Acme.jar:./objectdraw.jar:. SnakeController WIDTH HEIGHT DELAY

[cs11fxyz@ieng6-201]:pa8$
```

4. Invalid height only

```
[cs11fxyz@ieng6-201]:pa8$ java -cp ./objectdraw.jar:./Acme.jar:. SnakeController 200 50
50
Error: height value 50 is out of range. It should be between 200 and 800
Usage: java -cp ./Acme.jar:./objectdraw.jar:. SnakeController WIDTH HEIGHT DELAY

[cs11fxyz@ieng6-201]:pa8$
```

5. Invalid height and delay

```
[cs11fxyz@ieng6-201]:pa8$ java -cp ./objectdraw.jar:./Acme.jar:. SnakeController 600 1 1
Error: height value 1 is out of range. It should be between 200 and 800
Error: height value 1 is not divisible by the segment size
Error: delay value 1 is out of range. It should be between 50 and 600
Usage: java -cp ./Acme.jar:./objectdraw.jar:. SnakeController WIDTH HEIGHT DELAY

[cs11fxyz@ieng6-201]:pa8$
```

6. Invalid everything

```
[cs11fxyz@ieng6-201]:pa8$ java -cp ./objectdraw.jar:./Acme.jar:. SnakeController 6034 37
1
Error: width value 6034 is out of range. It should be between 200 and 800
```

```
Error: width value 6034 is not divisible by the segment size
Error: height value 37 is out of range. It should be between 200 and 800
Error: height value 37 is not divisible by the segment size
Error: delay value 1 is out of range. It should be between 50 and 600
Usage: java -cp ./Acme.jar:./objectdraw.jar:. SnakeController WIDTH HEIGHT DELAY

[cs11fxyz@ieng6-201]:pa8$
```

You need to include the newline between the usage statements and the prompt.
If this is not matched EXACTLY, NO partial credit will be given for these test cases.

2 – Setting up the GUI:

Implement all the UI of the game in **`SnakeController.java`**.

Buttons, Labels, and Layout

- A "Score: " label that records the score of current game.
  - The label is placed on the upper left corner of the game board
  - Upon start, the initial score is 0. So the the label should show "Score: 0"
  - The score increases by 10 every time the snake eats a piece of food.

- A "High Score: " label that records the highest score obtained so far
  - The label is placed on the upper right corner of the game board
  - Upon start, the initial high score is 0. So the label should be "High Score: 0"
  - The high score will only update when you press new game after the game ends (whether it be on game over or on a win) AND the new score obtained is greater than the old high score.

- A "New Game" button that resets the game board to the initial state
  - The button is centered on the lower panel of the game board
  - If the button is clicked, the "Score" label should be reset to 0, the value on the "High Score" label should be updated, and the snake should be reset to its initial size (more later).

- The rest of the main frame will be the space where the snake is moving around, and where food can be placed (i.e., the canvas)

- After initializing all the labels and buttons, call validate to reset the canvas size. (**Note:** The canvas dimensions might be a few pixels off sometimes. It's just how the MainFrame is laid out.)

Creating the Snake and Food

- The snake will have a light grey head (Color.LIGHT_GRAY, 50 x 50 pixels) and a white body composed of white FilledRects (Color.WHITE, 50 x 50 pixels each). Initially, the snake is only the head.
- The snake should be created as close to the middle of the canvas as possible at the start of each game (the snake must be on a divisible by 50 boundary at all times). The following is the initial start location for snake:

```
snakeX = xOffset + ( ( windowWidth / 2 ) / SEG_SIZE ) * SEG_SIZE;
snakeY = ( ( windowHeight / 2 ) / SEG_SIZE ) * SEG_SIZE;
```

Note: The snake will always start at the same default location when the game begins.

Note: Yes the 2's above count as magic numbers--define constants for them (and don't name the constant "TWO").

- Food should be placed randomly on the board, but it should never be placed on top of the snake. The food will be a 50 x 50 pixels orange FilledOval (Color.ORANGE).

Resizing

- This application **does not** support canvas resizing. Canvas resizing will not be tested.
- The canvas must be initialized as a valid size (for more info, see the command line arguments section in Part 3 Implementation).

Boundaries of Game Board

- You will also need a FramedRect that defines the boundaries of the game board. This is to clearly define boundaries since the canvas size sometimes won't be the same dimensions passed into the command line. The offset of the frame can be found using the following formula:

```
xOffset = ( canvas.getWidth() - windowWidth ) / 2;
```

  Where windowWidth is the width of the window that was passed in on the command line. Also, make sure that this value is always greater than or equal to 0. Again, the 2 here is a magic number.

3 – Snake Functionality:

Implement all of the Snake's functionality in **`Snake.java`**.

When the game starts, the snake and piece of food will be on the board (see Creating the Snake and Food for where to place them). Once any key is pressed, the snake will begin to move. The default direction is left. If an arrow key is pressed first, then the snake will start moving in that direction. If the space key is pressed first, then the game will pause.

After the game begins, the snake will move in response to the four arrow keys. Depending on what arrow key is pressed, the associated direction will be set as the Snake's current direction. When the snake is just the head, then the snake can move in any direction. Once the snake eats food (there is a body), then it will be unable to move in the opposite direction to which it is currently moving. This is to prevent the snake from moving over itself.

Each time the snake eats food, it grows by 1 block (which is appended to the end of the body). Growing should be handled by the Snake. The snake also needs to signal the controller to increment the score. It must also request the controller to place a new food on the canvas. The game is over when a certain amount of food has been placed.

If the space key is pressed any time the snake is moving, then the pause text should be shown/hidden in the controller, and the snake should pause movement. The direction that the snake is moving should not be changed while the game is paused. This means that if the snake was moving to the right when the game was paused, then no matter how many times we press the arrow keys, the snake should continue to move right when the game is unpaused.

4 – Implementation:

Note:

1) DO NOT use any data structures that have not been covered in class (HashMap, HashTable, HashTag, HashAnything). You **must** only use ArrayList.

2) DO NOT use any static variables to communicate between classes. (only constants can be static)

Direction Utility

Direction.java (provided)

This is a utility class that encapsulates the possible directions a snake could be traveling. For the purposes of this assignment, you will only need to use the enum to select one of the four directions, and use it to compare directions. More specifically, you can access these directions in your code like this: `Direction.UP`, `Direction.DOWN`, etc. If you want to compare these directions, use the == operator. This class is provided in the starter code for this assignment.

The Controller

SnakeController.java

SnakeController needs to implement the KeyListener and ActionListener interfaces and provide an implementation for the keyPressed method. The keyReleased and keyTyped methods should have an empty implementation.

At the end of begin and actionPerformed, add this line of code:

```
canvas.requestFocusInWindow();
```

Without this line, you will need to click the canvas every time you want the KeyListener to work. This line of code makes sure that the focus is always set to the canvas. Normally If you were to click the New Game button, the focus would shift to the button (that's why when a button is normally selected / highlighted, you can press space to click the button). However, in this assignment, we don't want this to happen, so adding this line of code avoids this issue.

The controller should also have JLabel objects that will be displayed once certain events occur. When the snake crashes, the gameOverText should be displayed to the screen (same with winText when a win occurs and pauseText for a pause). These text objects should be added to the bottom panel to the right of the button.

Aside from initializing UI elements, the controller is also responsible for placing food on the canvas, and updating the text objects that show the game's state (paused, game over, win). This first task becomes complicated, because we must place food in an open space. In other words, we cannot place food on top of the snake. This will be handled in the `placeNewFood` method.

public static void main(String[] args):
Once you have determined that the command line arguments are all valid, then you can create a new Acme.MainFrame in main. Because we need access to all the command line arguments as non-static variables, we are going to change the SnakeController constructor to take in a String [] args, instead of just using the default constructor.

```
SnakeController game = new SnakeController( args );
new Acme.MainFrame( game, game.windowWidth,
                    game.windowHeight + Y_PADDING);
```

The parameter passed into the Acme.MainFrame call sets the size of the entire frame, and placing other components such as labels and buttons on this frame will reduce the size of the canvas. To avoid this, we'll pad the vertical dimension argument with a 50 pixel padding. There's been a weird error when I tested this on ieng6. Sometimes, the width of the canvas is 6 more pixels that what was specified in the argument list. Print the canvas width after your call validate, and check to see if it equals what was passed in. If this error is consistent, edit the padding. You must remove this print statement before the turning in your code. It is only there to help you adjust your canvas size properly.

public void keyPressed(KeyEvent e):
This method should listen to the space bar and the arrow keys. Make sure to use the Direction utility class mentioned above to keep track of the directions the snake is traveling in.

If the **space key** is pressed, toggle the value of the paused boolean variable, and either display or hide the pause text on the screen, depending on the value of the boolean. The direction the snake is moving should be preserved when the game is paused. If the snake is moving left, then it should still move left when the game is unpaused.

The **arrow keys** will determine the direction the snake should move next. Whenever an arrow key is pressed, we should set the direction of the snake. If the snake is moving in one direction and has a body, and the user pressed the key that represents the opposite direction, the snake should not eat itself (*hint*: you should check this when you move the Snake). In other words, instead of eating itself, the snake should just keep moving in the direction it was moving before the key was pressed.

You will need to get the keyCode from the event to determine which key was pressed. For the space key and the arrow keys, you will have to use virtual key codes. Eg: to check for a press of the left arrow key:

```
int keyCode = e.getKeyCode();
if( keyCode == KeyEventVK_LEFT ) {
    // do some stuff
}
```

When the game opens, pressing any key (other than the arrow keys) should start the game, with the snake moving in the default direction (left). Pressing any of the four arrow keys should also start the game, with the snake moving in the direction corresponding to the arrow key that was pressed.

When the game ends, whether it be from a win or a loss, the movement keys will not do anything. On a win or loss, the **only** way to reset the game is to press the "New Game" button. You should be able to click the "New Game" button at **any point** of the game (including when the game is paused). Only when the game is properly reset (snake head is placed in the correct location and a fruit is randomly placed) can the game start by pressing one of the directional keys (or any key on the keyboard). This will cause the snake to start moving in the direction of the arrow key that was pressed (or left if a non-arrow key was pressed).

In addition to responding to the movement keys, the snake must also be able to pause its movement. The pause action will occur when the user presses the space key. If the user presses pause, the snake needs to ignore all incoming input from the keyboard, except for another space key press. For example, if the snake is moving down, and the user presses space, followed by up, and then space again, the snake should still be moving down. Pausing should also make the pauseText appear on the canvas (likewise, unpausing should make the pauseText disappear).

public Location placeNewFood()

Food can only be placed in a randomly selected open space. We will be randomly generating locations similar to how we generated them in PA5. We first generate random x and y coordinates, then check if those coordinates overlap with the snake. Remember that the x and y coordinates you generate must be aligned on the invisible grid that the snake travels on. If they are contained in the snake, then we will generate new random numbers and recheck (what kind of loop works best?). Once we have valid coordinates, we will move the food to this new location. Remember to return this location as well.

We must also increment the score each time we place a new food except the first time that we place food (this is so that we can count how many food pieces have been generated).

Additionally, we will need to keep track of how much food we have placed on the canvas. This means that we will need to calculate the maximum amount of food we can place (*hint*: this will be equal to the number of 50px squares on the canvas - 1, since the head takes up one square by default). The maximum value of the food should be calculated in the constructor, after the arguments are parsed.

Once we generate the maximum amount of food, then we will signal victory and stop the execution of the snake.

The Snake

Snake.java

This class defines what a Snake is (an ArrayList of SnakeSegments and a head SnakeSegment), and what a Snake can do (it is an Active object that moves in a fixed direction, as determined by the controller). The snake's constructor must have this signature:

```
        public Snake( DrawingCanvas canvas, Location start, int delay,
                      SnakeController controller )
```

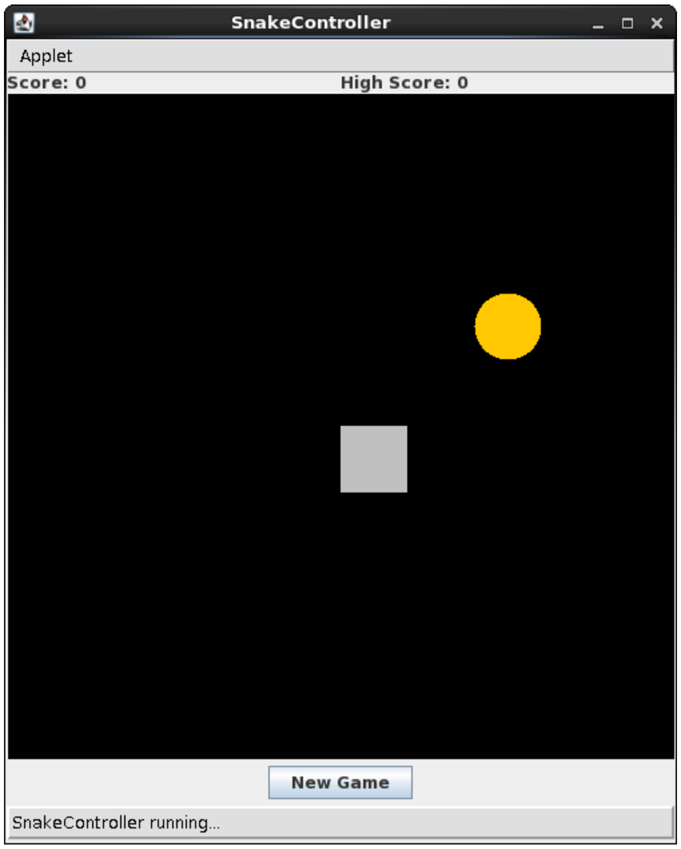
This time, we need to pass in the controller into the Snake constructor because the snake will need to inform the controller if the snake has crashed, and that the game is over. We also need to ask the controller to place new food, and check if the snake is within the bounds of the canvas.

SnakeSegment
This private inner class is defined within Snake, and defines what a segment of a snake should be. A SnakeSegment is going to need a FilledRect to represent its visual appearance on the canvas. Each SnakeSegment should also have a Location of where it is on the grid. This class should be able to return its Location through a getter, and should be able to move itself (hint, implement a moveTo method for SnakeSegment, in addition to a move). You should also ensure that any methods that affect the SnakeSegment are declared as `synchronized`.

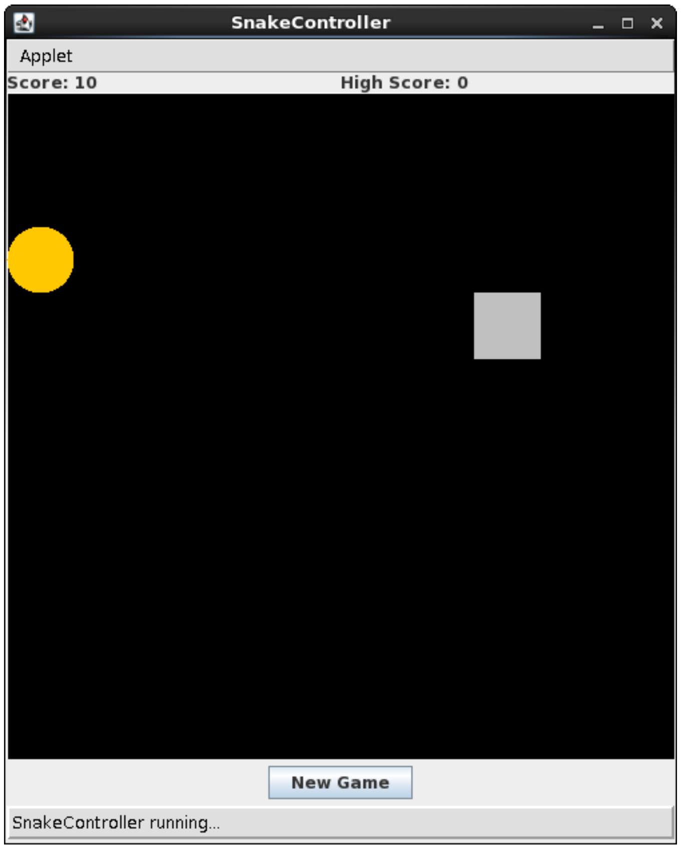
private synchronized void move(dx, dy)
Move the head by (dx, dy), and keep track of where the old location was. We then move the snake body one piece at a time to the location of the preceding piece. You should also check that the new movement direction that was set by the controller, if any, is valid. If the direction is invalid, then the snake should keep moving in the direction that it was moving before the invalid direction was set. Otherwise, the snake should begin moving in the new direction.

private synchronized void grow()
This method will add a new snake segment to the end of the body. The new segment should be placed at the same location as the last element in the snake. Be mindful that the first time the snake grows, this first body segment will be added at the same location as the head, since the body will be empty at this point.

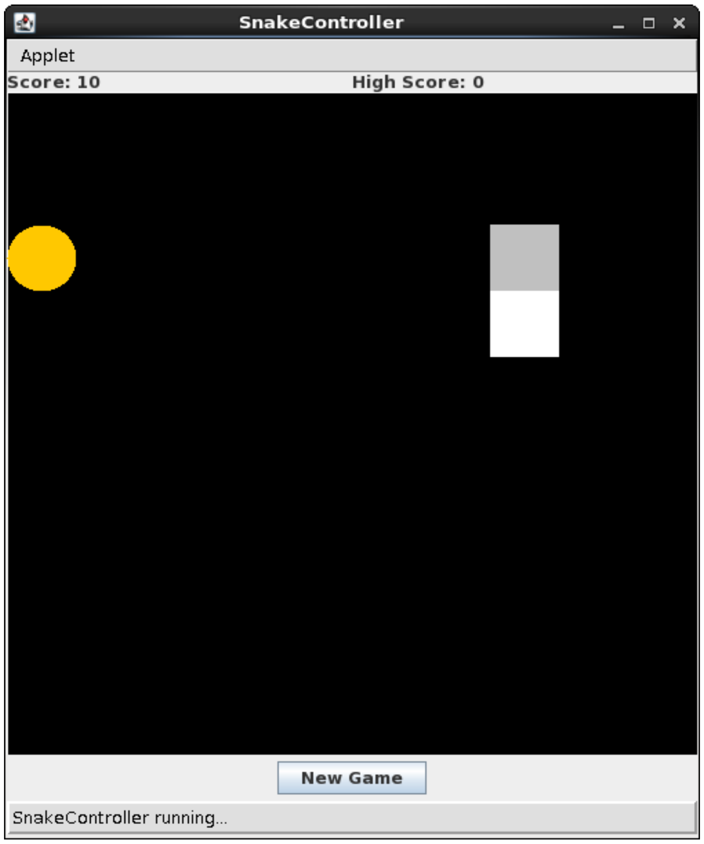
5 – Sample Screenshots:



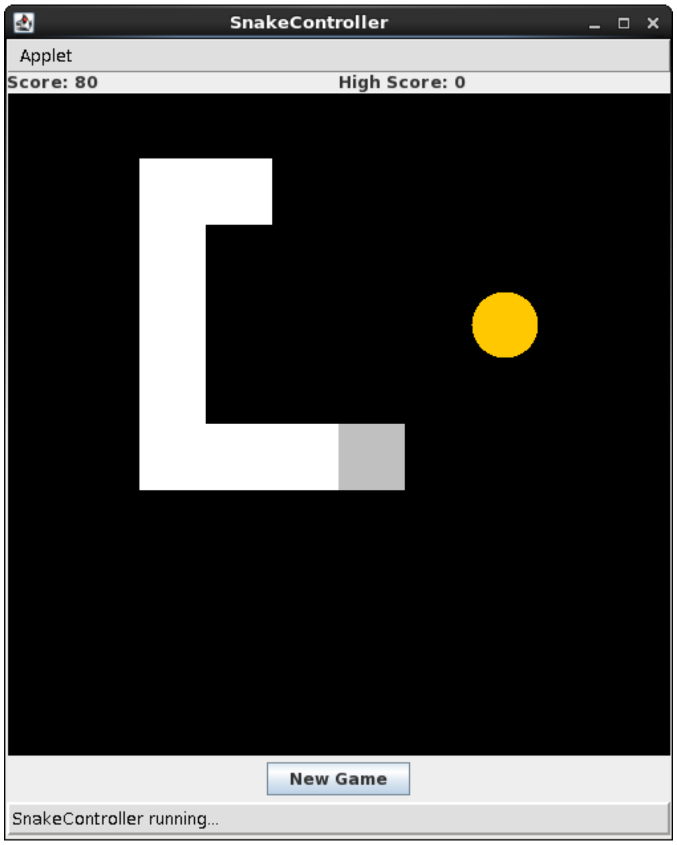
Game at start up (canvas size is 500 x 500).



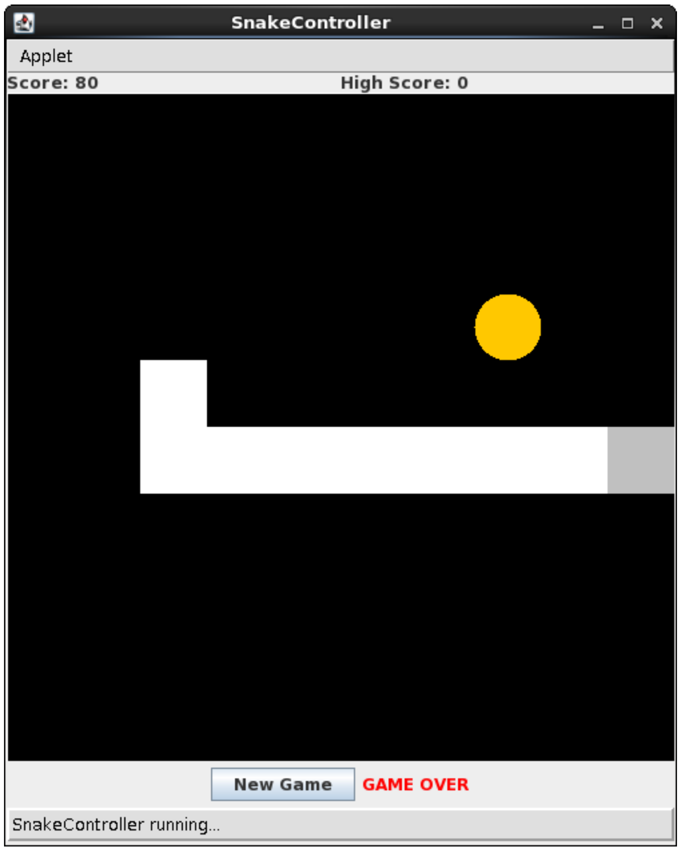
Snake eats food, score updates, new food is placed.



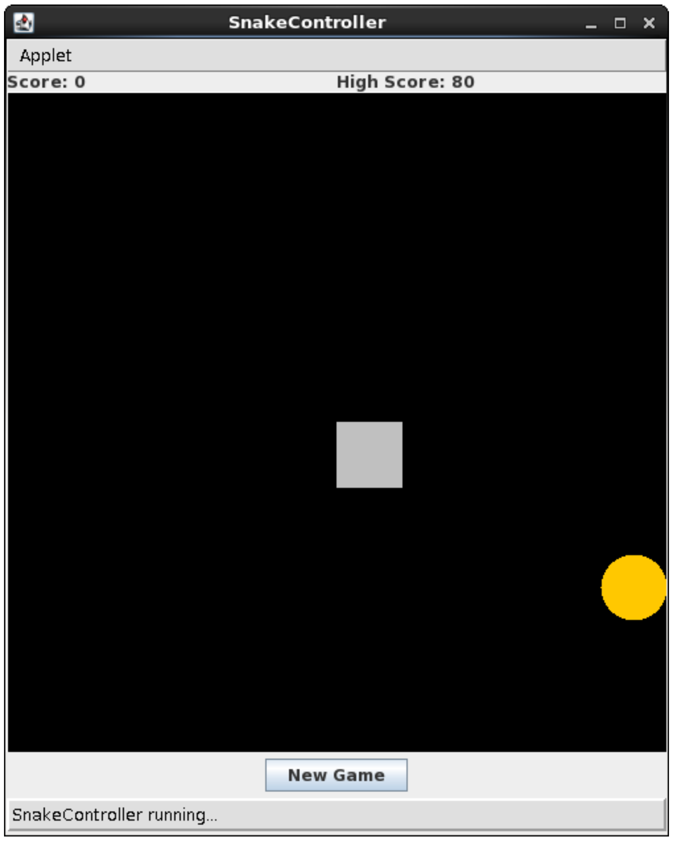
Snake grows by one segment after consuming food.



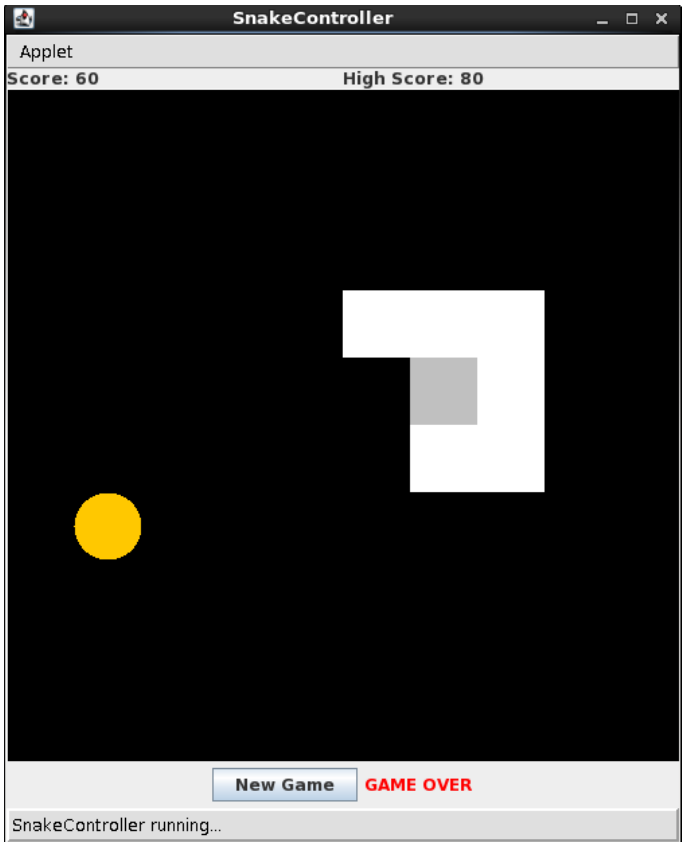
Snake after a few iterations of eating food.



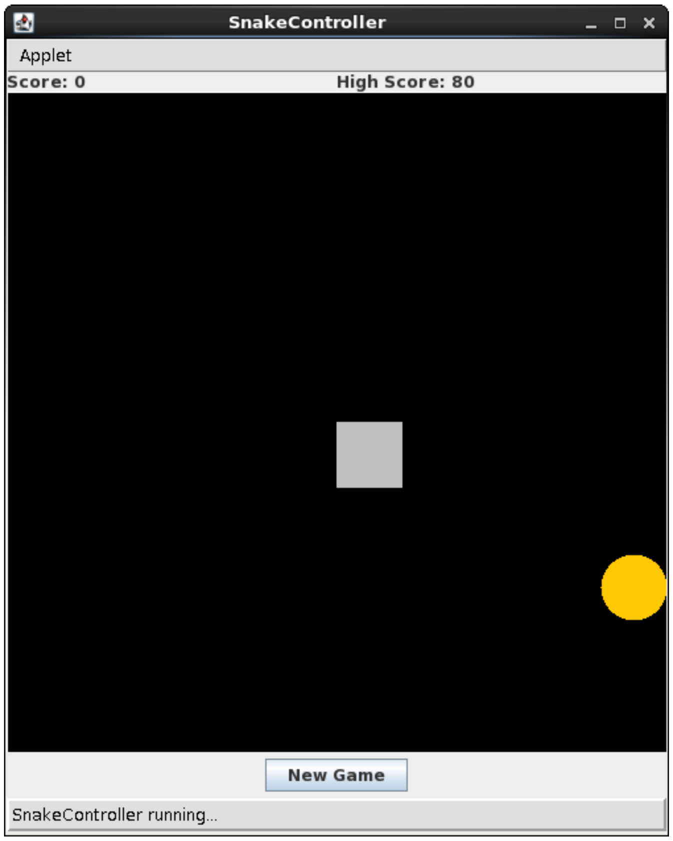
Game Over when Snake hits wall.



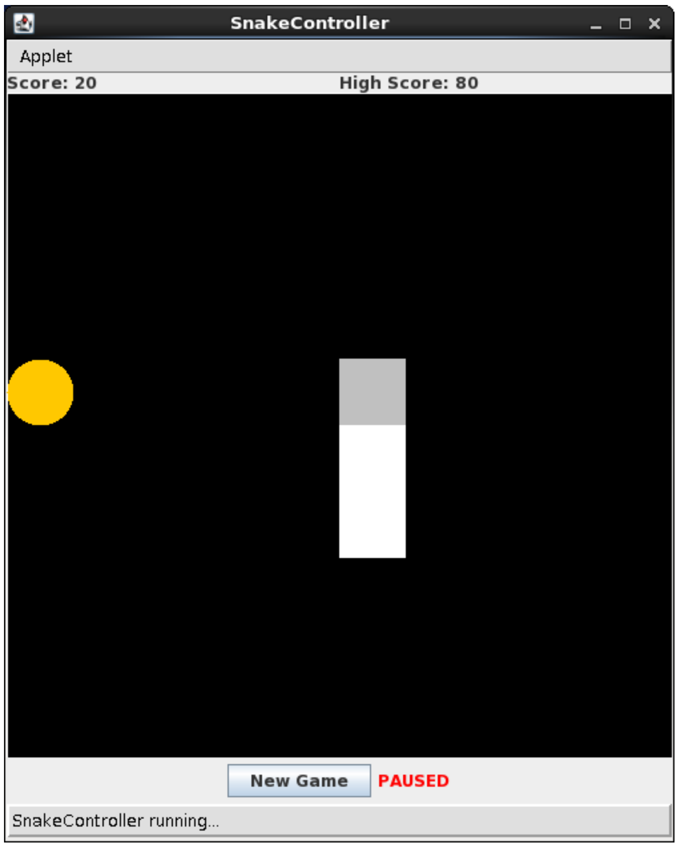
High Score of the game is updated.



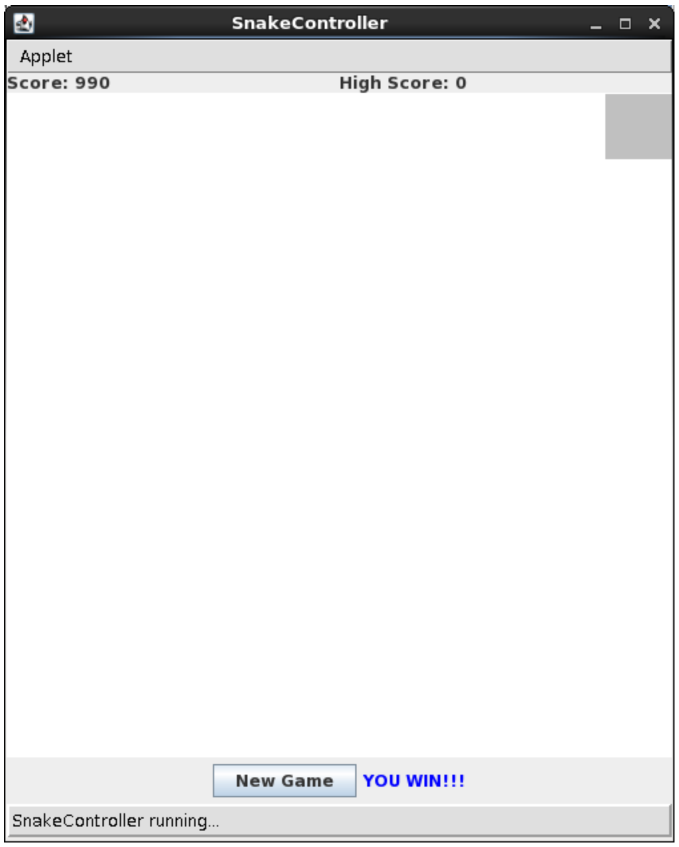
Game over when Snake runs into itself.



High Score remains the same.



Game is paused.



Winning the game!

WARNING: MUST READ

DO NOT start late.

This will be a long, difficult assignment. There will be NO mercy for those that start late.

EXTRA CREDIT (5 points)

For this extra credit, create two new files: EC_SnakeController.java and EC_Snake.java.

If you do not make these extra credit files, 1) you will not get extra credit and 2) your file will be graded as if that is your regular file. TL;DR: Make separate EC files or your grade will suffer!

```
> cd ~/pa8
> cp SnakeController.java EC_SnakeController.java
> cp Snake.java EC_Snake.java
```

1) Changing color of snake to match fruit [1 point]

Have the snake start off with just the head. The fruit's color, however, should be randomly selected from this array:

```
Color [] FOOD_COLORING = { Color.RED, Color.ORANGE, Color.GREEN,
                           Color.BLUE, Color.YELLOW };
```

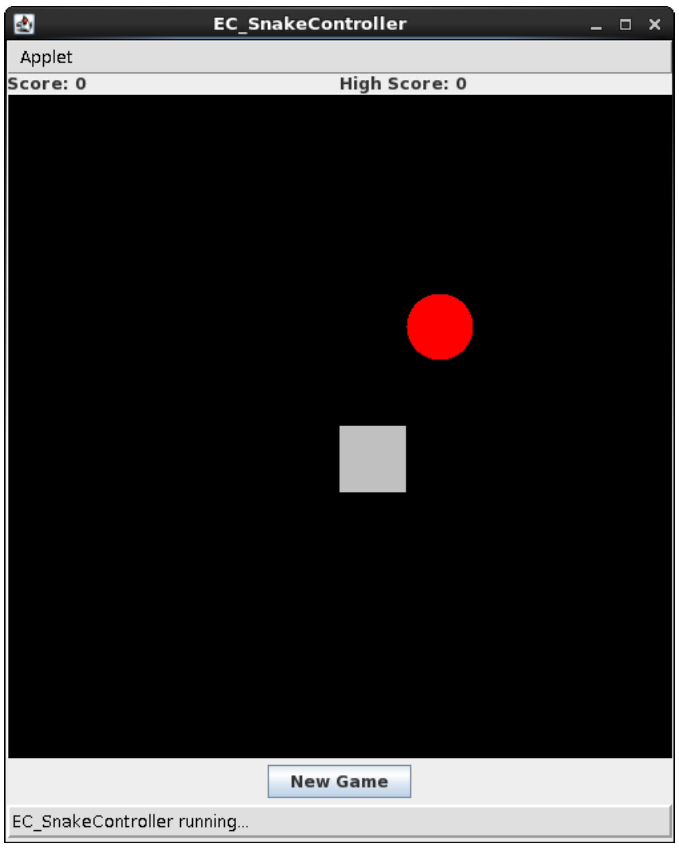
Each time the snake eats a food, the body of the snake should change color to match the color of the food it just ate. Note that the head should always remain the same light gray color that it had during the regular assignment.

2) Snake eat Self (Cut Off Mode) [2 points]

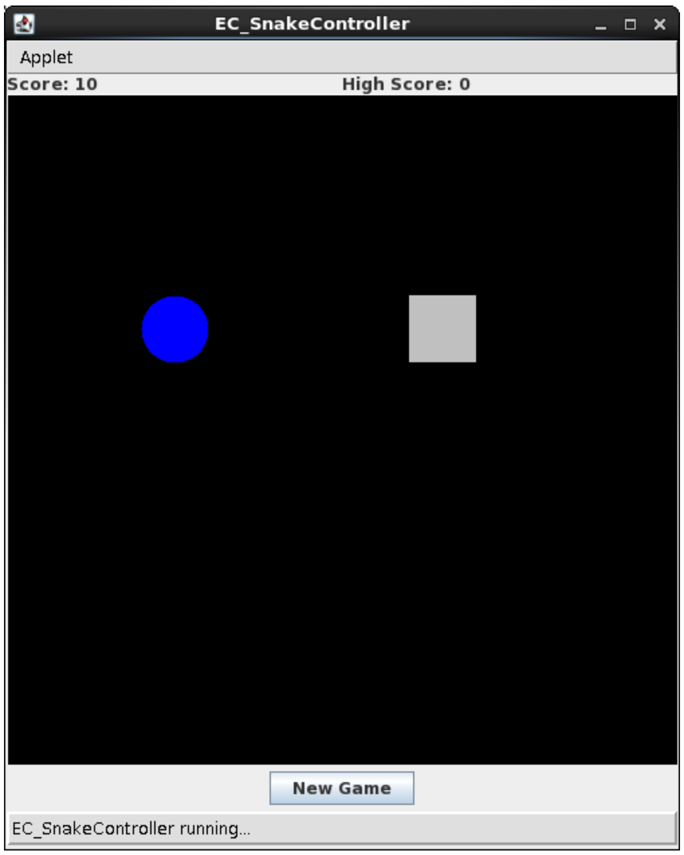
In this game mode, the snake learns a cool trick from its reptilian cousin, the lizard. The snake loses it's tail if it bites itself. Gameplay works as normal (as in, crashing into the walls results in game over). The only change we will make for this mode is that you no longer die when the snake bites itself.

To implement this game mode you will need to find out which segment the snake ate. Since we maintain an arraylist for the snake body, we can find the index at which the head comes in contact with the body. Once the correct index has been found, all elements from that index to the end of the snake should be removed from the arraylist and the canvas. Each segment lost should make a deduction from the total points. A player loses 10 points per segment lost.

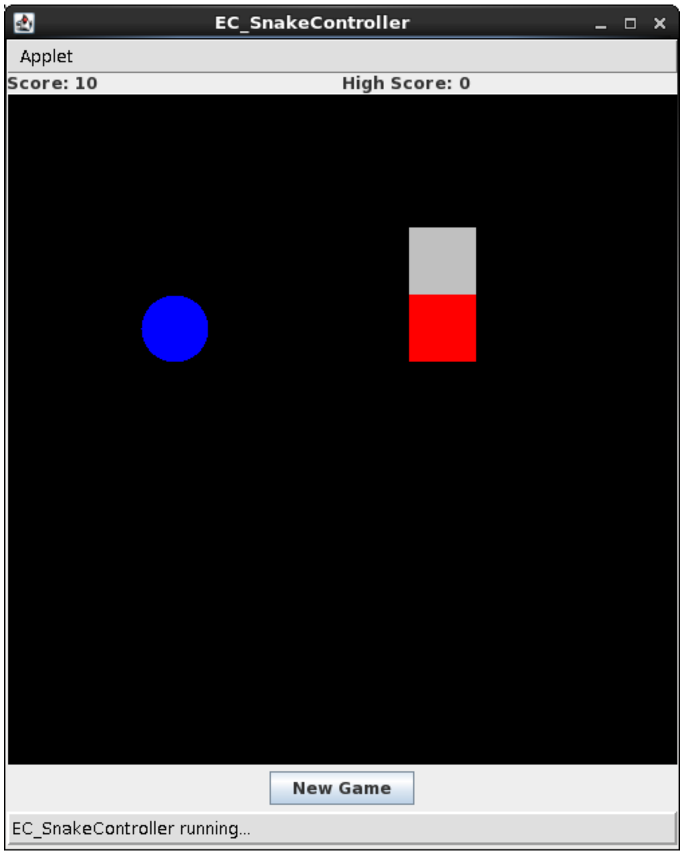
<u>**Sample Screenshots:**</u>



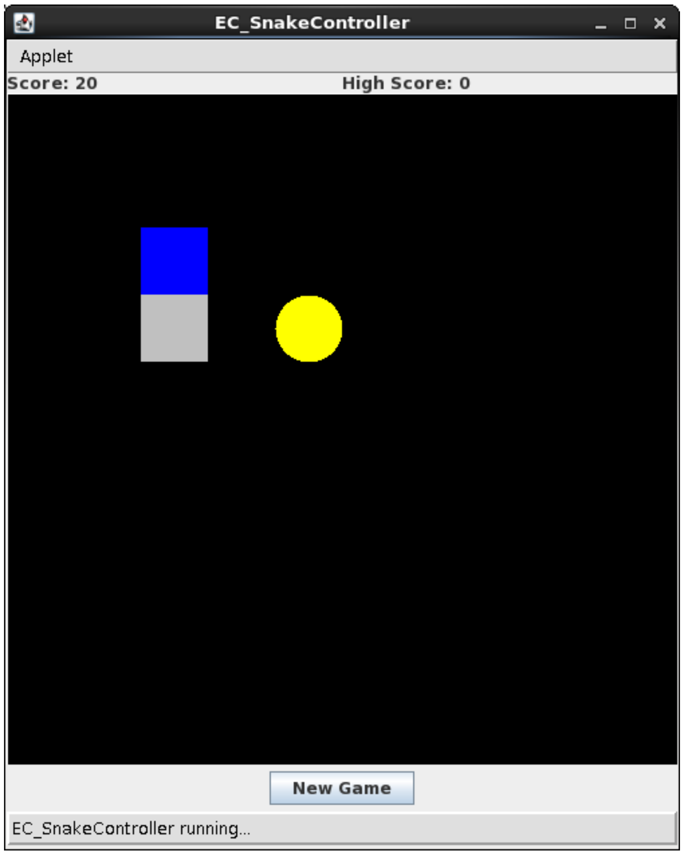
Game at begin.



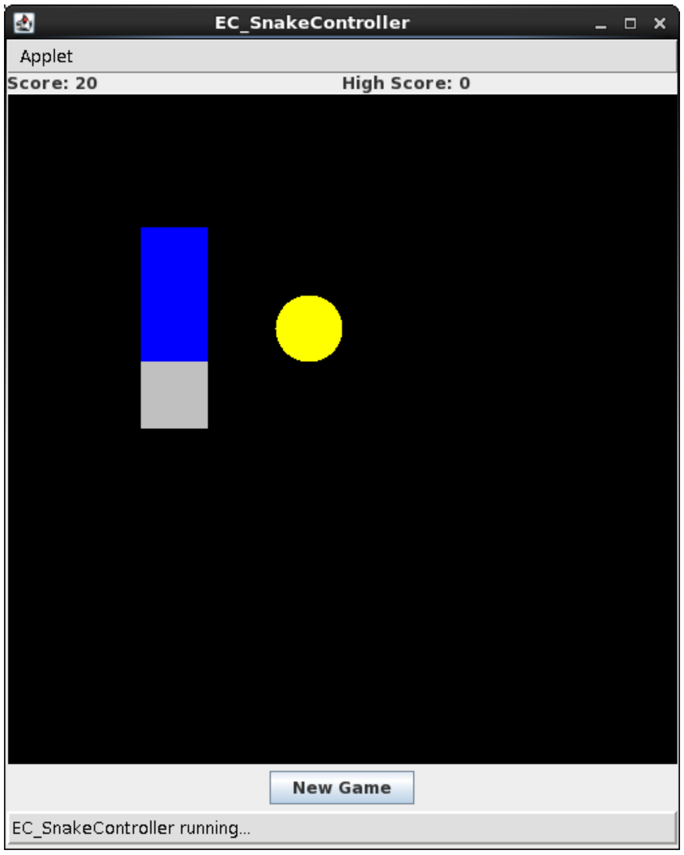
Snake eats food, score updates, new food is placed.



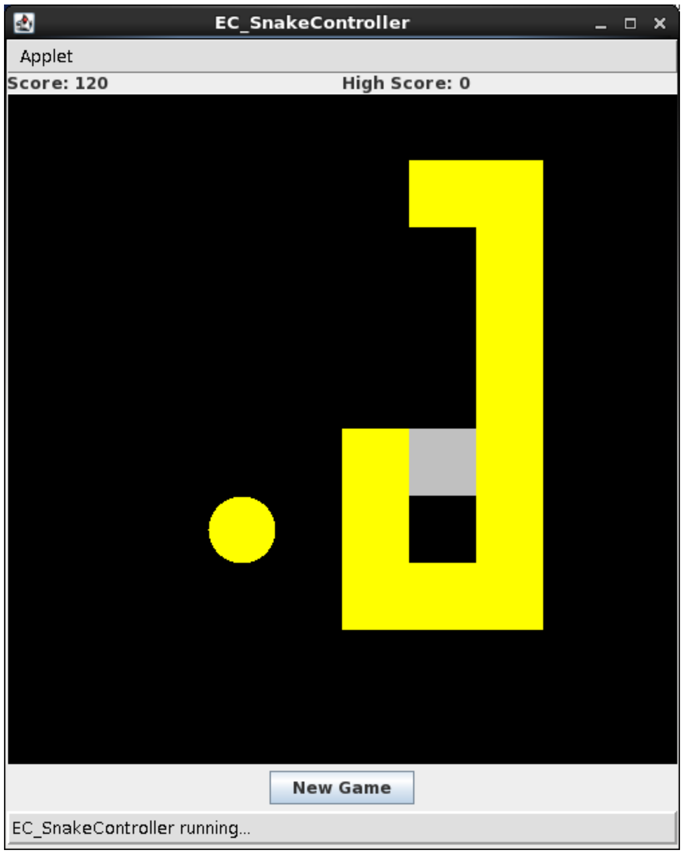
After eating a red food, the snake grows and is red.



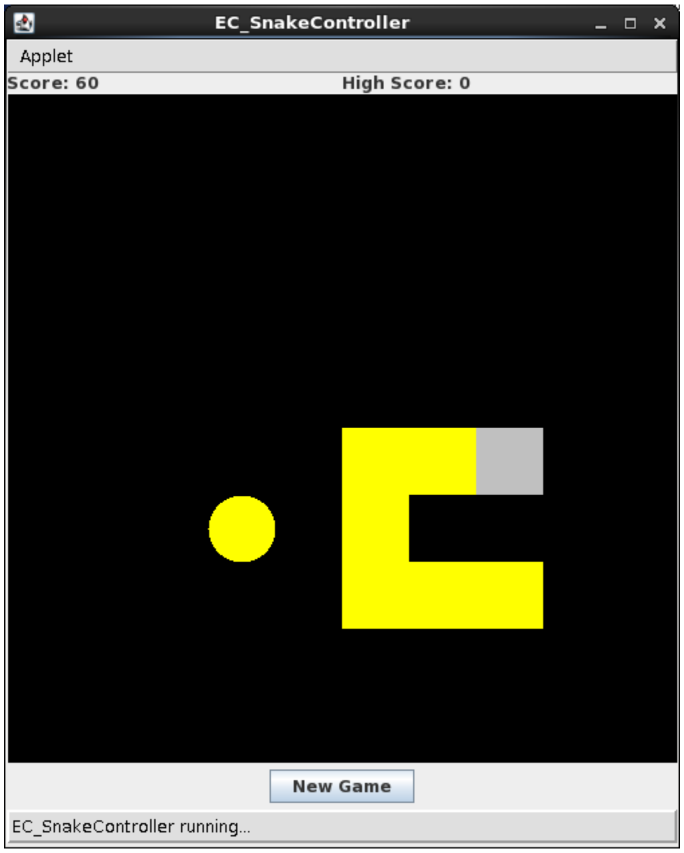
Snake eats blue food, and immediately turns blue.



After eating food, snake grows by one segment



Snake before consuming self, notice the current score



Snake after eating self, score decreases as segments are cut off

3) Early Turnin [2 points]

You will be given points based on turning in before the due date.

- 2 points for turning in 48 hours before the due date (or earlier).
- 1 point for between 24 hours and 48 hours before the due date.

We will only count the last submission you make, so be careful to test your code before turning in to avoid any last minute bugs.

Turnin

To turnin your code, navigate to your home directory and run the following command:

```
> cse11turnin pa8
```

You may turn in your programming assignment as many times as you like. The last submission you turn in before the deadline is the one that we will collect.

Verify

To verify a previously turned in assignment,

```
> cse11verify pa8
```

If you are unsure your program has been turned in, use the verify command. **We will not take any late files you forgot to turn in.** Verify will help you check which files you have successfully submitted. It is your responsibility to make sure you properly turned in your assignment.

Files to be collected:

README
Snake.java
SnakeController.java
Direction.java
PA8Strings.java
Acme.jar
objectdraw.jar

Additional files to be collected for Extra Credit:

EC_Snake.java
EC_SnakeController.java

The files that you turn in must have EXACTLY the same names as those above.