Part 1: Program Specification

What will your program look like at the end?

- When the program is run, a graphics window should pop up and prompt the user to presses the spacebar to start the two-player snake game
- Gameplay Rules:
    - Each snake will try to eat as many edible items (pieces of food) as it can.
    - With each additional edible item consumed, the snake grows longer.
    - The game ends either when a snake eats itself, eats another snake, runs into the side of the screen, eats more than 3 inedible item or when the user prompts the game to end.
    - As the game progresses, the arena background changes color once the players reach specific scores. The snakes change color based on keyboard input.

How will the user input work? (e.g., clicking, keys on the keyboard, specifying a file, what have you)

- Keyboard keys (WASD and arrow keys) to move the snake, (C and ?) to change snakes' color

How will the program respond?

- The snake changes direction based on the input from the keyboard. Depending on the player, the W or Up Arrow keys make the snake move up, the S or Down Arrow keys make the snake move down, etc.
- Pressing L prompts the game to end
- For the WASD user, pressing C causes the snake to change color; for the Arrow key user, pressing the ? key changes the color of the snake

What purpose does it serve? (e.g., is a game, a productivity tool, a screen saver?)

- The program is a game

Classes

1. Class Arena extends JPanel implements Colorable
2. Interface Colorable
3. Class Item (imports java.awt.image)
    a. Class Apples extends Item
    b. Class Oranges extends Item
    c. Class Rocks extends Item
    d. Class iPhones extends Item
4. Class Snake implements Colorable
5. Class Segment extend Snake
6. HISS

Arena

Semantics:

- Extends JPanel and implements Colorable
- Creates the arena in which the game is played
- Only one instance is created

Member Variables:

public static final int WIDTH, HEIGHT

- Semantic representation: Defines the graphics window
- Why public and final: These member variables are public and final because they should be accessible from the Snake and Item classes, and these variables cannot be changed.
- Why static: Even though only one instance of Arena is created, we want these member variables to apply to the entire class and not just to one instance of Arena; therefore, these member variables should also be static.
- Why int: WIDTH and HEIGHT are ints because in this context we have no need for a floating decimal point

public int score

- Increases every time a snake consumes an edible item
- Sum of the length of the two snakes
- Why public: needs to be accessible from class Snake

Constructors:

- One constructor that takes no arguments because class Arena is only used to create the window of gameplay using predefined member variables.

Methods:

public void paintComponent (Graphics g) { … }

- Draws the arena and overrides the existing imported paintComponent constructor
- Displays the score of the game
- Why public: should be accessible from class HISS
- Why non-static: should be accessible from a non-static context because an instance of Arena is created in HISS
- Non-recursive

public void changeColor() { … }

- Overrides the method from interface Colorable
- Changes the color of the Arena when a specific score is reached
- Why public: needs to be accessible from class *HISS*

Colorable (Interface)

Semantics:

- Used to change the color of snakes and the arena (background of the screen)
- When the score reaches a certain level, the background color changes
- Users can change the color of their snake with keyboard inputs

Member variables:

public Color colorSnake1, colorSnake2, colorArena

- Pre-specified colors
- Why public: needs to be accessible from other classes like Snake and Arena

Methods:

public void changeColor()

- No body
- Why public: will be over-ridden in class Arena and Snake

Item

Semantics:

- Imports java.awt.image
- 4 items are placed randomly in the Arena
- There are 4 instances of Item at a time (2 edible Items and 2 inedible Items)
- Once a snake eats an edible Item, that particular edible Item disappears and another Edible Item appears randomly in the Arena
- Once a snake eats an inedible Item, that particular inedible Item disappears and another inedible Item appears randomly in the Arena. If a snake eats more than 3 inedible Items, the snake dies

Member Variables:

public int positionX, positionY

- Semantic representation: Each item has a x and y coordinate in the Arena
- Why public: We want to access the position of an item in class Snake
- Why int: we do not have the need for the position of an item to be a floating decimal point

public final int width, height

- Semantic representation: each item has a width and a height
- Why public: we want to access the width and height of an item within class Snake
- Why int: we do not have the need for the width and height of the item to be a floating decimal point

public boolean edible

- Semantic representation: some items are edible and some are not
- Why public: needs to be accessible from Snake class
- Why boolean: item can either be edible or inedible, so a boolean would serve the purpose

Constructors:

- One constructor with no arguments
- Width and height are defined specifically in the constructor
- positionX and positionY are randomly determined within the constructor

Methods:

public void drawItem () { …}

- Draws item on the screen
- Randomizes type of edible Item (Apples, Oranges)
- Randomizes type of inedible Item (Rocks, iPhones)
- Why public: Should be accessible from class Arena
- Non-recursive

public void eraseItem () { ... }

- Erases edible item when the snake eats it
- Why public: Should be accessible from class Arena
- Non-recursive

Snake

Semantics:

- Defines behavior of the snakes
- Two instances are created: one for each snake

Member Variables:

private double positionX, positionY

- Semantic representation: Each snake has a position. The position of the snake refers to the position of the head of the snake
- Why double: The snake should be able to move along the screen smoothly, so floating point decimals are needed
- Why private: we do not need to access the positions of snakes in other classes

private double velocityX, velocityY

- Semantic representation: Each snake has a velocity
- Why double: So that the snake can move more smoothly on-screen
- Why private: we do not need to access the velocity of snakes in other classes

public int length

- Semantic representation: Each snake has a length. The length increases by one when one edible item is consumed
- Why int: The length of a snake is determined by how many pieces of food it has. Because each piece of food can't be split up, this variable is an int.

- Why public: we need the sum of the lengths of the two snakes to determine the score achieved in the game. We will access the lengths of the two snakes in class Arena

private int inedibleCount
- Semantic presentation: If a snake eats more than 3 inedible items it will die
- Why private: doesn't need to be accessed anywhere else other than class Snake

public ArrayList body
- Semantic representation: the amount of edible items a snake eats or has initially is what makes up a snake's body. Multiple Segments make up a snake. When a snake eats an edible item, the snake grows another Segment
- Why ArrayList<Segments>: In order to represent the fact that a snake is made up of multiple Segments, the snake's body should be represented by an arraylist of Segments.
- Why private: we do not need to access the snake's body in other classes

private Color color
- Represents the color of the snake
- Why private: doesn't need to be accessed from other classes specifically

Constructors:
- One constructor, no arguments
- The positions of each snake should start in the middle of the Arena
- Velocity of each snake is also specified in the constructor
- The length of a snake is initialized in the constructor
- The body of each snake is also created in the constructor, and it depends on the snake's initial length

Methods:
public void update(double time) { … }
- Will be called while game is running to move the snake around
- Why public: needs to be accessible from class HISS

public void draw() { … }
- Draws the snake to the screen
- Why public: needs to be accessible from class Arena

public void changeDirection (char c) { … }
- Reads character pressed from the keyboard and determines the direction in which the snake will move

- Why public: Needs to be accessible from class *HISS*

public boolean eatItem(ArrayList<Item> items) { … }
- Checks to see if the snake has eaten the item
- Why public: needs to be accessible from class *HISS*

public void evolve() { … }
- length of snake increases by 1 if edible item is consumed
- Inedible count increases by 1 if inedible item is consumed
    - If inedible count == 3, snake dies
- Why public: needs to be accessible from class *HISS*

public boolean eatSelf () { … }
- Checks if the snake is eating itself
- Why public: needs to be accessible from class *HISS*

public boolean eatFriend () { … }
- Checks if the snake is eating its friend
- Why public: needs to be accessible from class *HISS*

public boolean hitWall () { … }
- Checks if the snake is hitting the wall
- Why public: needs to be accessible from class *HISS*

public void changeColor(char c) { … }
- Overrides the change color method from interface Colorable
- Iterates through an arraylist of colors to change the color of the snake when the user presses a key on the keyboard
- Why public: needs to accessible from HISS

Segment

Semantics:
- Represents each segment of the snake
- Extends Snake

Member Variables:
public double positionX, positionY

- Semantic representation: Each segment has its own position which overrides from parent class Snake
- Why double: The snake should be able to move along the screen smoothly, so floating point decimals are needed
- Why public: needs to be accessible from class Snake

public int width, height
- Semantic representation: represents the width and height of each segment
- Why public: needs to be accessible from class Snake

Constructors:

public Segment(double posX, double posY)
- One constructor taking two arguments
- Width and height will be declared and initialized outside the method

Methods:
- No methods needed for Segment; it is only used to make up the body of the snake

HISS

Semantics:
- Entry point into the game (To run game, run java HISS)
- No instance exists
- Extends KeyboardListener

Member Variables:

public char c
- Used to receive keyboard input

public static final int FPS
- Semantic representation: Used to measure time in the program
- Why public and final: FPS should be accessible from the Snake class and cannot be changed
- Why static: it defines class HISS and not just an instance of class HISS
- Why int: FPS is an int because in this context we have no need for a floating decimal point

Constructors:

No constructors because no instance is created

Methods:

public void main (String[] args) { … }

public void run () { … }

- Runs the game
- Called in the main method to run the game itself

Interactions between Classes

1. java HISS is used to run the game
    a. HISS creates an instance of **Arena**
        i. Arena creates two instances of **Snake** and four instances of **Item**
2. Play game!