3. WRITTEN RESPONSES

3 a.

3.a.i.

The overall purpose of the program is to control a snake that lives to eat food. The player controls the snake, eating food that increases the size of the snake. If the player loses control of the snake and either runs into the walls or runs into themselves, the game ends, revealing the player's final score.

3.a.ii.

The video demonstrates how the game is played, including eating food, moving around, and crashing.

3.a.iii.

The input demonstrated in the video is keyboard key presses, and the outputs demonstrated in the video are the controls to the game, the snake's growth, the randomization of the food, the movement of the snake, the crashing of the snake, and the final score is shown after the snake crashes.

3 b.

3.b.i.

```
match self.direction {
    Direction::Left => {
        if self.overlap_tail(self.position.x - 1, self.position.y) {
            self.is_alive = false;
            return;
        }
        self.position.x -= 1;
        self.tail.pop_back();
        self.tail.push_front(self.position);
    }
    Direction::Right => {
        if self.overlap_tail(self.position.x + 1, self.position.y) {
            self.is_alive = false;
            return;
        }
        self.position.x += 1;
        self.tail.pop_back();
        self.tail.push_front(self.position);
    }
    Direction::Up => {
        if self.overlap_tail(self.position.x, self.position.y - 1) {
            self.is_alive = false;
            return;
        }
        self.position.y -= 1;
        self.tail.pop_back();
        self.tail.push_front(self.position);
    }
    Direction::Down => {
        if self.overlap_tail(self.position.x, self.position.y + 1) {
            self.is_alive = false;
            return;
        }
        self.position.y += 1;
        self.tail.pop_back();
        self.tail.push_front(self.position);
    }
    Direction::Still => {}
}
```

3.b.ii.

```
pub fn draw(&self, c: &Context, g: &mut G2d) {
    draw(
        Color::SNAKE_HEAD,
        self.position.x as u32,
        self.position.y as u32,
        1,
        1,
        c,
        g,
    );
    for seg in self.tail.iter().skip(1) {
        draw(Color::SNAKE_BODY, seg.x as u32, seg.y as u32, 1, 1, c, g);
    }
}
```

3.b.iii.

The name of the list being used is tail.

3.b.iv.

The data contained in the list represents the coordinates of the snake's tail.

3.b.v.

The list helps manage the complexity of the code by allowing me to draw the snake's tail on the screen effortlessly. If I did not use the list, there would be no way for me to accomplish this. I would have to use a list that can change its size depending on the size of the screen and the size of the snake's tail. If the player increases their screen size, allowing for the snake to grow longer, the list would also have to grow. The list allows the code to still function even after changes in the game; it would be impossible to code the game without the list.

3 c.

3.c.i.

```
pub fn spawn(&mut self, size: Size, snake: &Snake) {
    loop {
        let mut thing: bool = true;
        for i in snake.tail.iter() {
            if i == &self.position {
                thing = false;
                break;
            } else {
                thing = true;
            }
        }

        if thing {
            break;
        } else {
            self.position = Position {
                x: thread_rng().gen_range(0..=(size.width / (BLOCK_SIZE * 2.0)) as i32),
                y: thread_rng().gen_range(0..=(size.height / (BLOCK_SIZE * 2.0)) as i32),
            };
        }
    }
}
```

3.c.ii.

```
pub fn update(&mut self, size: Size, args: &UpdateArgs, key: Key) {
    self.snake.update(size, args.dt, self.key_direction(key));
    self.window_size = size;
    if self.snake.position == self.food.position {
        self.snake.eat();
        self.food.spawn(size, &self.snake);
    }
}
```

3.c.iii.

The spawn function will generate a random coordinate that will not conflict with the snake's location and its tail. This function contributes to the program's overall functionality by allowing food to be spawned in a location that doesn't kill the snake.

3.c.iv.

The spawn function takes in 3 parameters: the food structure, the window size that the snake game is played in, and the snake structure. The spawn function then loops while the food coordinates are the same as any part of the snake's body. Inside the loop, the food's data element 'position' will get updated by generating a random integer for the X and a random integer for the Y part of the coordinate. The range for the random number for the X is inclusive and is from 0 to the width of the screen divided by the product of the size of the snake's head times two. The range for the random number for the Y is also inclusive and is from 0 to the height of the screen divided by the product of the size of the snake's head times two.

3 d.

3.d.i.

First call:

The first call to the function occurs when the program is first ran; it takes in the size of the screen and the snake. This function gets ran resulting in a random coordinate that the food will be spawned to.

Second call:

Later throughout the code, whenever the player eats the food, the function's input is changed since the snake changes. The new output is another random coordinate for which the food will be spawned to, but the number of possible coordinates gets lessened by one since the snake is longer now (longer by one coordinate). Another possible difference in the function getting called again is if the size of the game is changed (enlarged or shrieked), the number of possible coordinate locations gets changed respectably.

3 d.ii.

Condition(s) tested by first call:

The condition tested in the first call is if the random coordinate of the food is valid, meaning if it is out of bounds.

Condition(s) tested by second call:

The conditions tested in the second call are if the food location is out of bounds and if the food is in conflict with the current location of the snake (meaning if the food is spawned on the snake).

3.d.iii.

Results of the first call:

The result of the first call is a random coordinate that fits into the default parameters of the snake and screen size.

Results of the second call:

The result of the second call is a new random coordinate that fits into the new parameters of the snake (the new length of the snake and the new location of the snake and its body) and the new parameter of the screen size. The result of the second call varies from user to user since it is entirely random.