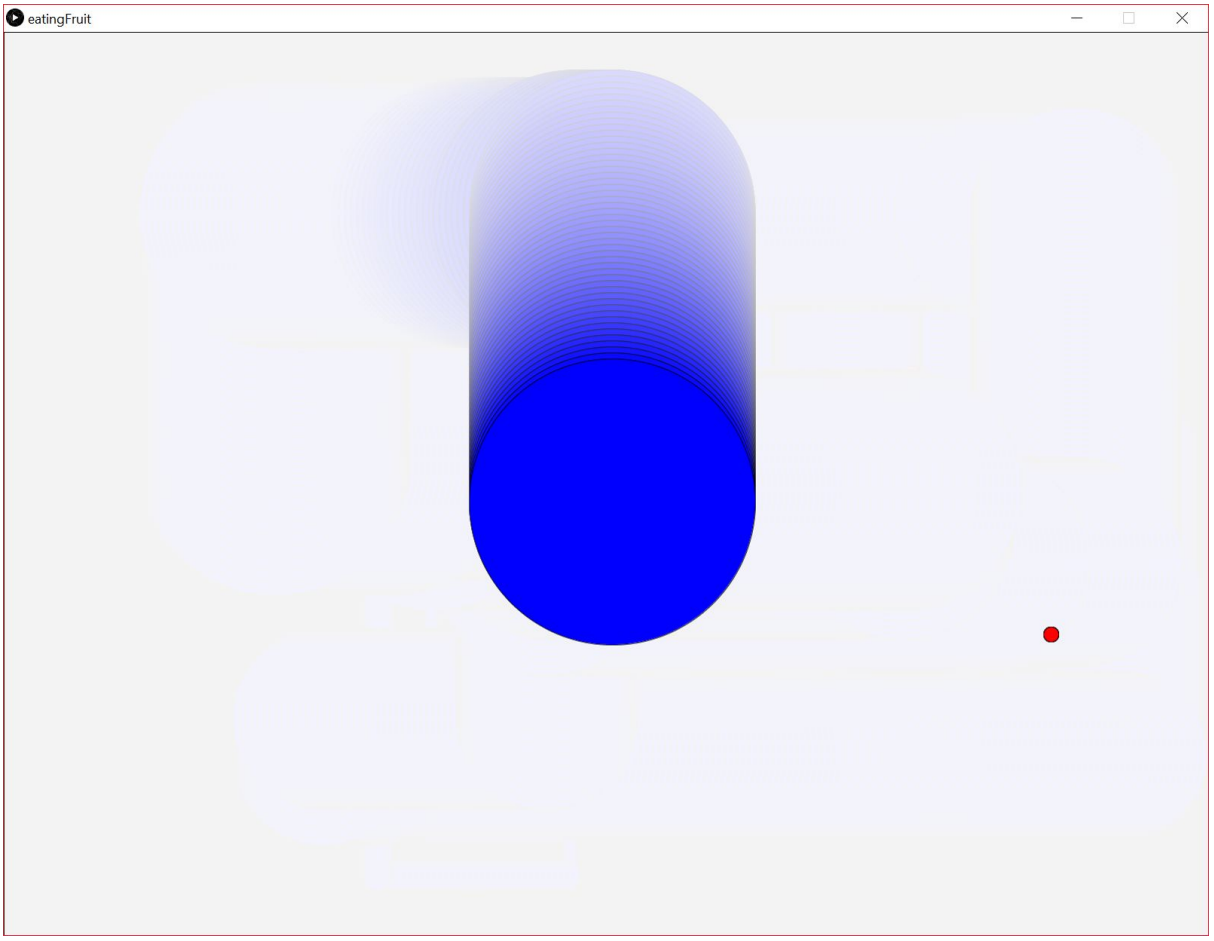
Eating Fruit

Beta Version

This game is based on snake, where the player controls a character (the snake) who has to eat as much fruit as possible before getting too big to fit on the screen. Each time the snake eats a piece of fruit you get points, but also get bigger. Once you touch the edge of the screen the game is over! The snake is always moving - the player can only change which direction.



Focus

The most important things to focus on while writing the code for this game:

- Use well-named variables, as much as possible.
- Remember rules for capital letters. Variable names are written in *lowerCamelCase* (notice the first letter is a lowercase ‘L’). Class names are written in *UpperCamelCase* (notice the first letter is an uppercase ‘U’).
- Most code should be split out into separate *functions*. You won’t just use an ellipse to draw the player - you will use a drawPlayer() function, and inside that function will be an ellipse.

- Concentrate on making the code as *readable* as possible. It should be easy to tell what all your code does (using clear variable names and function names helps with this)
- ADVANCED:
    - Once you have built the basic game you will use this as a starting point to practice writing *classes*. You will, at the very least, have a *Player* class and a *Fruit* class. Remember that the code for the *Player* class will go in the *Player* tab, likewise for *Fruit* in the *Fruit* tab.

NB: This is an advanced lesson! You will need to figure out what variables you need, alongside lots of other bits - like remembering to put in void setup()! Some code is provided, but it's up to you to figure out the rest.

Starting Point

The void draw() of the basic game (before writing classes) will look something like this:

```
void draw() {
  fill(255, 10);
  rect(0, 0, width, height);
  fill(0);
  if (!gameOver) {
    drawScore();
    drawPlayer();
    movePlayer();
    checkCollision();
    drawFruit();
  } else {
    gameOver();
  }
}
```

The simplest function, drawPlayer(), is declared like this (remember that this happens *after* the end of void draw()):

```
void drawPlayer() {
  fill(0, 0, 255);
  ellipse(playerX, playerY, playerRadius*2, playerRadius*2);
}
```

This is the *starting point* for movePlayer() - note that you will need to add controls for the other directions:

```
void movePlayer() {
  playerX = playerX + playerSpeedX;
  playerY = playerY + playerSpeedY;
  if (keyPressed) {
    if (key == 'w') {
      playerSpeedX = 0;
      playerSpeedY = -playerSpeed;
    }
  }
}
```

The collision detection works like this:

```
void checkCollision() {
  if (dist(playerX, playerY, fruitX, fruitY) < (playerRadius + fruitRadius)) {
    score = score + 1;
    fruitX = random(playerRadius, width - playerRadius);
    fruitY = random(playerRadius, height - playerRadius);
    playerRadius = playerRadius + 10;
  }
  if (playerX > width - playerRadius || playerX < playerRadius ||
    playerY > height - playerRadius || playerY < playerRadius) {
    gameOver = true;
  }
}
```

Not the positioning of fruitX and fruitY when there is a collision. Why do we subtract playerRadius from width?

Other tips

- We have used a transparent rectangle instead of a background - this allows us to have the trail that you see in the screenshot at the start.
- Most variables should be declared as *float* data types. gameOver should be a *boolean.*
- It's best to make score an *int* - but you can try it as a *float* to see the difference.
- resetGame() is a function which resets every variable you have to its starting position. It sets *gameOver* to *false*, *score* to 0, etc.
- Once you have made the basic game you can make the player and the fruit look much cooler using more shapes, adding images, etc. The benefit of the drawPlayer() and drawFruit() functions is that all those changes will happen with those functions.
- ADVANCED:
    - When creating your *Player* and *Fruit* classes, remember that all functions to do with the player will go in the *Player* class (such as drawPlayer() and

movePlayer()). The variables to do with the player (such as playerX, playerXSpeed, playerRadius, etc) will also go in there.

- There are many places you could put the collision detection at this point. For simplicity, best to leave it in the main tab where it was before you implemented classes. You can access variables such as *playerX* by using the dot notation. If you've called your *Player* object *myPlayer*, then accessing *playerX* will be done through *myPlayer.playerX*.