sketch_200127a

```
//Declare an ArrayList with Stone and Gold objects.
ArrayList<Stone> stones = new ArrayList<Stone>();
ArrayList<Gold> golds = new ArrayList<Gold>();

//Declare a Player object.
Player player = new Player(300,50);

//Declare a Score object.
Score score = new Score(10, 10);

PImage bgimage;
boolean setupphase;

//Sets program windowsize to 600x600 px.
void settings() {
 size(600,600);
}

//Runs setup on the program. Loading and setting background image. Adds the specific Stone and Gold objects to the ArrayList.
void setup() {
 frameRate(60);
 bgimage = loadImage("../sprites/soil.jpg");
 background(bgimage);

 setupphase = true;

 //Adds minerals to ArrayList
 stones.add(new Stone(int(random(width)), int(random(150, 350)), 0));
 stones.add(new Stone(int(random(width)), int(random(150, 350)), 1));
 stones.add(new Stone(int(random(width)), int(random(150, 350)), 2));
 stones.add(new Stone(int(random(width)), int(random(150, 350)), 3));
 stones.add(new Stone(int(random(width)), int(random(150, 350)), 4));

 golds.add(new Gold(int(random(width)), int(random(300, height)), 0));
 golds.add(new Gold(int(random(width)), int(random(300, height)), 1));
 golds.add(new Gold(int(random(width)), int(random(300, height)), 2));
 golds.add(new Gold(int(random(width)), int(random(300, height)), 3));
 golds.add(new Gold(int(random(width)), int(random(300, height)), 4));

}

//Main program loop
```

```
void draw() {

 //Setupphase loop; places the mineral object until no minerals intersect.
 while (setupphase) {
  for (int i = stones.size()-1; i >= 0; i--) {
   for (int j = golds.size()-1; j >= 0; j--) {
    Stone sto = stones.get(i);
    Gold gol = golds.get(j);

    //While 2 different minerals intersects > run regen() function, which adds new random coordinates.
    while (overlap(sto.x, sto.y, gol.x, gol.y, sto.radius, gol.radius)) {
     regen();
     break;
    }
   }
  }
  //When nothing intersects break out of Setupphase loop
  setupphase = false;
  break;
 }

 //If the user restarts the program by pressing &apos;q&apos; > run setup to get new specific objects.
 if (player.restart && stones.size() + golds.size() == 0) {
  setup();
 }

 //Game loop;
 //1. Set background to bgimage
 background(bgimage);

 //2. Invoke Player basic methods for rendering and moving.
 player.display();
 player.update();
 player.movement();

 //3. Invoke Score methods
 score.display();

 //4. Iterate over all the Stone and Gold objects to invoke rendering methods and invoking Player grap
method for all Stone and Gold objects.
 for (int i = stones.size()-1; i >= 0; i--) {
  Stone sto = stones.get(i);
  sto.display();
   player.grap(sto.mineralCollision(player.x2, player.y2)[0],sto.mineralCollision(player.x2,
```

```
player.y2)[1]); //mineralCollisonNumber[0] = number in array, mineralCollisonNumber[1] = type
 }

 for (int j = golds.size()-1; j >= 0; j--) {
   Gold gol = golds.get(j);
   gol.display();
   player.grap(gol.mineralCollision(player.x2, player.y2)[0], gol.mineralCollision(player.x2,
player.y2)[1]); //mineralCollisonNumber[0] = number in array, mineralCollisonNumber[1] = type
 }

 //5. Iterate over all Stone and Gold objects to check if they should be removed from ArrayList, because
they were caught.
 for (int i = stones.size()-1; i >= 0; i--) {
   Stone sto = stones.get(i);
   if (sto.caught) {
     stones.remove(i);
   }
 }

 for (int j = golds.size()-1; j >= 0; j--) {
   Gold gol = golds.get(j);
   if (gol.caught) {
     golds.remove(j);
   }
 }

 //Ending loop;
 //When the user has caught all minerals > show ending screen; final score and instruction for restarting.
 if (stones.size()+golds.size() == 0) {
   background(0);
   textAlign(CENTER, CENTER);
   fill(#FFFFFF);
   textSize(32);
   text("You earned: " + score.money + " this game.", width/2, height/2);
   textSize(26);
   text("Press &apos;q&apos; to restart", width/2, height/2+36);
 }
}

//Function for replacing Stone and Gold obejcts. Using ArrayList.set instead of ArrayList.add to replace
existing ArrayList values.
void regen() {
 stones.set(0, new Stone(int(random(width)), int(random(150, 350)), 0));
 stones.set(1, new Stone(int(random(width)), int(random(150, 350)), 1));
```

```
 stones.set(2, new Stone(int(random(width)), int(random(150, 350)), 2));
 stones.set(3, new Stone(int(random(width)), int(random(150, 350)), 3));
 stones.set(4, new Stone(int(random(width)), int(random(150, 350)), 4));

 golds.set(0, new Gold(int(random(width)), int(random(300, height)), 0));
 golds.set(1, new Gold(int(random(width)), int(random(300, height)), 1));
 golds.set(2, new Gold(int(random(width)), int(random(300, height)), 2));
 golds.set(3, new Gold(int(random(width)), int(random(300, height)), 3));
 golds.set(4, new Gold(int(random(width)), int(random(300, height)), 4));
}

//Function for checking if intersectiong between 2 objects.
boolean overlap(float p1x, float p1y, float p2x, float p2y, float p1r, float p2r) {
 if (dist(p1x, p1y, p2x, p2y) < p1r + p2r) {
   return true;
 } else {
   return false;
 }
}

//Registers if a key is pressed and sends it to Player object.
void keyPressed() {
 player.setMove(key, true);
}

//Registers if a key is released and sends it to Player object.
void keyReleased() {
 player.setMove(key, false);
}
```

Mineral_gen_class

```
class Mineral {

 int worth;
 int weight;
 int x;
 int y;
 int radius;
 PImage sprite;
 int n;
 int type;
 boolean caught;

//Constructor sets start values for position, dimensions, number and if caught.
```

```
Mineral(int xpos, int ypos, int numberInArray) {
  x = xpos;
  y = ypos;
  worth = 0;
  weight = 0;
  radius = 0;
  n = numberInArray;
  caught = false;
}

//Method for displaying the object.
void display() {
  imageMode(CENTER);
  image(sprite, x, y, radius, radius);
}

//Method for when the Player intersects with a Mineral object creates an array with the specific objects index number and type.
int[] mineralCollision(float a, float b) {
  int[] mineralCollision = new int[2];
  if (dist(a, b, x, y) < radius) {
    mineralCollision[0] = n;
    mineralCollision[1] = type;
   }
   return mineralCollision;
 }

 //Method for changing a variable for determining if the object is caught and pulled back.
 boolean hasCaught() {
   return caught = true;
 }
}

//New type of Mineral; Gold. Extends the Mineral class meaning it has all of the same methods and variables as Mineral.
class Gold extends Mineral {

 //Load sprite image file.
 PImage g_sprite = loadImage("../sprites/gold.png");

 //Constructor changes a few variables.
 Gold(int xpos, int ypos, int numberInArray) {
   super(xpos, ypos, numberInArray);
   x = xpos;
```

```
  y = ypos;
  worth = 4;
  radius = int(random(10, 40));
  weight = radius*10;
  sprite = g_sprite;
  type = 1;
 }
}

//New type of Mineral; Stone. Extends the Mineral class meaning it has all of the same methods and variables as Mineral.
class Stone extends Mineral {

 //Load sprite image file.
 PImage r_sprite = loadImage("../sprites/rock.png");

 //Constructor changes a few variables.
 Stone(int xpos, int ypos, int numberInArray) {
  super(xpos, ypos, numberInArray);
  x = xpos;
  y = ypos;
  worth = 1;
  radius = int(random(25, 65));
  weight = radius*3;
  sprite = r_sprite;
  type = 2;
 }
}
```

Player_class

```
class Player {

 int x1, y1, x2, y2;
 boolean isUp, isDown, reset, restart;
 float theta;
 float thetaIncrease;
 float lineL;
 int lineIncrease;
 float r;

 //Constructor sets start values for position and rendering properties.
 Player(int xpos, int ypos) {
  x1 = xpos;
  y1 = ypos;
```

```
  theta = 0;
  thetaIncrease = 0.035;
  lineL = 25;
  lineIncrease = 4;
  r = 10;
}

//Part 1 of rendering; method for displaying the object.
void display() {
  line(x1, y1, x2, y2);
  ellipseMode(CENTER);
  fill(#b6b6b6);
  circle(x2, y2, r);
}

//Part 2 of rendering; method for updating the object, when user doesn&apos;t interact with program.
void update() {
  theta += thetaIncrease;
  if (theta > PI || theta < 0) {
    thetaIncrease *= -1;
  }
  x2 = int(x1+cos(theta)*lineL);
  y2 = int(y1+sin(theta)*lineL);

  if (lineL < 0) {
    theta += thetaIncrease;
  }
}

//Method for determining collision between Player and a mineral object.
void grap(int mineralCollisionNumber, int mineralType) {
  score.calcMoney(mineralCollisionNumber, mineralType); //Invoke calcMoney() for the intersecting
object to find amount of money to be added.

  //Iterate over all Stone objects;
  for (int i = 0; i < stones.size(); i++) {
    Stone sto = stones.get(i);
    if (mineralCollisionNumber == sto.n && mineralType == sto.type) {
      //Set the collided object to Player position;
      sto.x = x2;
      sto.y = y2;

      while (sto.y >= 70) { //Pull Player and collided object back to PLayer startposition;
        lineL -= lineIncrease;
```

```
      x2 = int(x1+cos(theta)*lineL);
      y2 = int(y1+sin(theta)*lineL);
      break;
    }
    if (sto.y < 70) { //When Stone object is pulled adequately back; inform user of which Stone has been caught, change caught variable to &amp;apos;true&amp;apos; to remove from the ArrayList in main program, add the money, reset Player to start values.
      println("Stone; " + sto.n + " got caught!");
      sto.hasCaught();
      score.money += score.moneyAdd;
      //Reset of Player
      pReset();
      break;
    } else {
      continue;
    }
  }
}

//Iterate over all Gold objects;
for (int i = 0; i < golds.size(); i++) {
  Gold gol = golds.get(i);
  if (mineralCollisionNumber == gol.n && mineralType == gol.type) {
    //Set the collided object to Player position;
    gol.x = x2;
    gol.y = y2;

    while (gol.y >= 70) { //Pull Player and collided object back to PLayer startposition;
      lineL -= lineIncrease;
      x2 = int(x1+cos(theta)*lineL);
      y2 = int(y1+sin(theta)*lineL);
      break;
    }
    if (gol.y < 70) { //When Gold object is pulled adequately back; inform user of which Gold has been caught, change hasCaught variable to &amp;apos;true&amp;apos; to remove from the ArrayList in main program, add the money, reset Player to start values.
      println("Gold; " + gol.n + " got caught!");
      gol.hasCaught();
      score.money += score.moneyAdd;
      //Reset of Player
      pReset();
      break;
    } else {
      continue;
```

```
    }
   }
  }
}

//Method for translating keyboard input to boolean values to determine what the user wants to do.
boolean setMove(char k, boolean b) {
  switch(k) {
   case &apos;w&apos;:
     return isUp = b;
   case &apos;s&apos;:
     return isDown = b;
   case &apos;r&apos;:
     return reset = b;
   case &apos;q&apos;:
     return restart = b;

   default:
     return b;
  }
}

//Method for changing the PLayer object based on setmove() output; based on keyboard input from the user.
void movement() {
  if (isUp) {
    lineL -= lineIncrease;
   } else if (isDown) {
    thetaIncrease *= 0;
    lineL += lineIncrease;
   }
   //If user wants to reset Player or Player returns to startposition then set the Player to start values.
   if (reset || dist(x2, y2, x1, y1) < 20) {
    pReset();
   }
}

//Method for resetting the Player to start values.
void pReset() {
  x1 = 300;
  y1 = 50;
  theta = 0;
  thetaIncrease = 0.035;
  lineL = 25;
```

```
  lineIncrease = 4;
  r = 10;
 }
}
```

Score_class

```
class Score {

 int money;
 int x,y;
 int moneyAdd;

 //Constructor sets start values for position and money.
 Score(int xpos, int ypos) {
  x = xpos;
  y = ypos;
  money = 0;
 }

 //Method for rendering the score at top left of game window.
 void display() {
  textAlign(LEFT, CENTER);
  fill(#FF79E7);
  textSize(28);
  text("Money: " + money, x, y);
 }

 //Method for calculating the amount of money to be added to money based on the specific caught
objects properties.
 int calcMoney(int mineralCollisionNumber, int mineralType) {
  if (mineralType == 2) {
   for (int i = stones.size()-1; i >= 0; i--) {
    Stone sto = stones.get(i);
    moneyAdd = int(sto.worth*sto.weight);
   }
  } else if (mineralType == 1) {
   for (int i = golds.size()-1; i >= 0; i--) {
    Gold gol = golds.get(i);
    moneyAdd = int(gol.worth*gol.weight);
   }
  }
  return moneyAdd;
 }
}
```