

Resumé

Dette projekt tager udgangspunkt i problemstilling om at sikre overlevelsen af spil bygget på Adobe Flash Player. Projektet rekreere spillet *Golder Miner* fra start 2000. Målet er at have en fungerende version af *Golder Miner* inklusive hovedfunktionerne af spillet.

Projektet ente ud i et program, der ligner det klassiske Gold Miner. Det har alle hovedfunktionerne med om at sænke en spiller arm ned gennem jorden og tage fat i mineraler, hvorefter at trække dem op og få penge af dem tilsvarende værdien af det specifikke mineral.

Programmet er kodet i Processing IDE v. 3.5.4 med sproget Processing, som er baseret på Java.

Indhold

1	Problemformulering	3
2	Funktionsbeskrivelse	3
3	Dokumentation	3
3.1	Spilletshistorie	3
3.2	Detaljeret beskrivelse af Mineral klassen og inheritance konceptet og brugen	4
3.3	Funktion og metode oversigt for GoldMiner	8
4	Test af spillet	12
5	Konklusion	14
6	Bilag - Flowcharts og source kode	15

1 Problemformulering

Eftersom Adobe stopper support til Adobe Flash Player i slutningen af 2020, vil en masse gamle Flash spil stoppe med at virke. Dermed rekreative jeg spillet Gold Miner, så man stadig kan spille det i fremtiden.

Det er i dag et problem, når firmaer, som Adobe, stopper med at understøtte software som Flash Player og lign. Det gør at ekstremt mange kreative ideer og tanker, bliver tabt til historien, da de stykker kode de bygger på ikke længere kan køres sikkert. Og man som hjemmesidevært ikke vil hoste spillene. Desuden vil hjemmesider som y8.com, miniclip.com og kongregate.com miste en stor del af deres spil katalog, hvorfaf langt de fleste af disse vil være gamle klassikere.

2 Funktionsbeskrivelse

Programmet er et spil og dermed bruges der brugerinput. Dette er i form af at brugeren bruger tastaturet, navnlig knapperne, *s* og *w* til at bevæge spilleren, og *r* og *q* for henholdsvis gå tilbage til startposition og genstarte spillet. Brugerens input baseres ud fra hans/hendes observation på skærmen. Spil vinduet er 600×600 px. Dette betyder at man kan spille spillet på siden, mens man laver noget andet, der ikke kræver ens fulde opmærksomhed. Desuden er de grafiske elementer scalert til at passe i et lille skærmvindue.

Programmet bruges som et middel for tidsfordriv og afslapning. Ligesom mange andre spil i samme stil, kræver spillet ikke den store koncentration. Der er ikke mulighed for multiplayer, hverken lokalt eller online.

3 Dokumentation

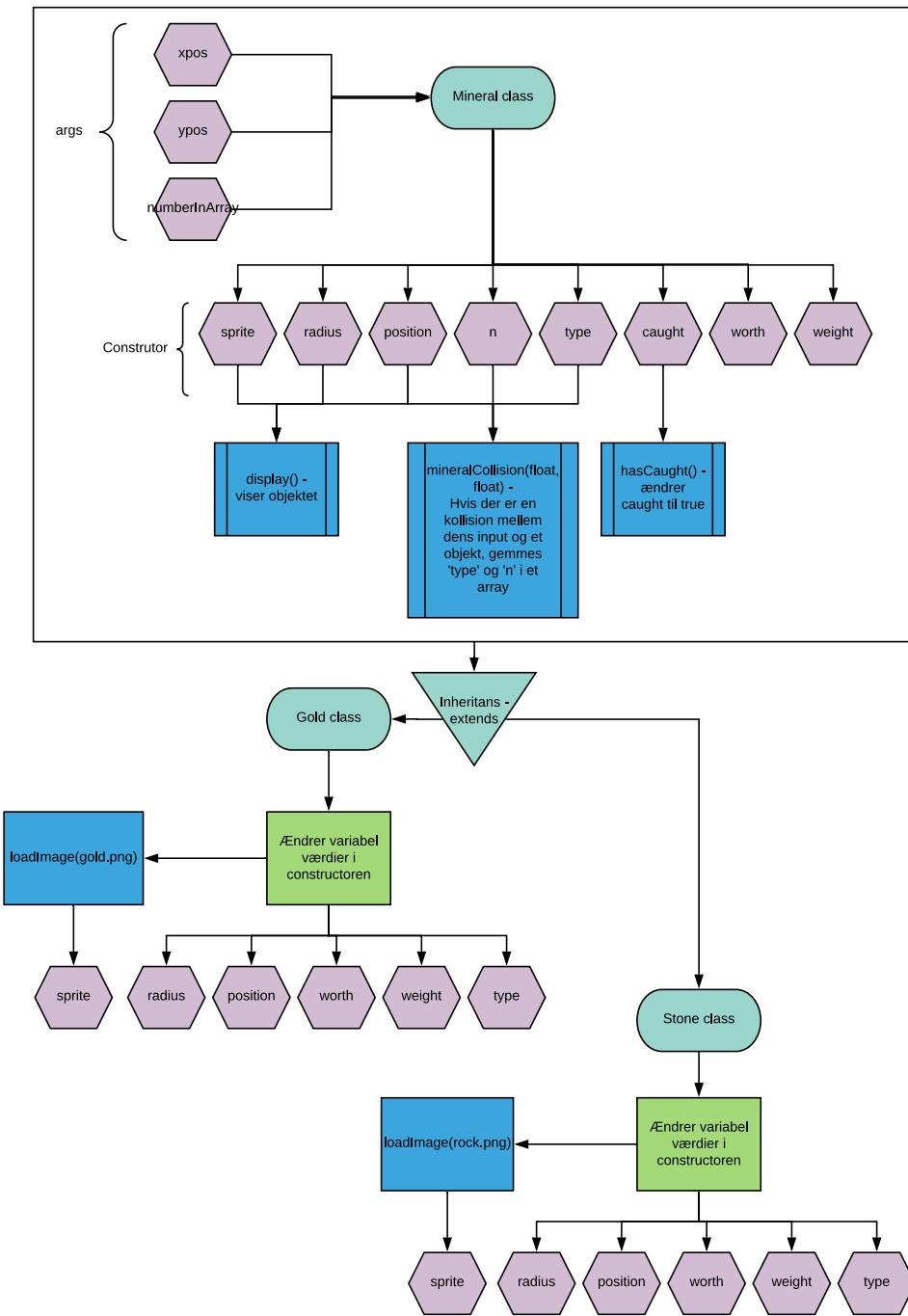
3.1 Spilletshistorie

Princippet i spillet bygger på at brugeren er en guld graver. Brugeren har en kran til at hive mineraler op fra jorden. Brugeren vil gerne tjene penge, og derved vil brugeren helst grave guld op, da man får flere penge af at grave guld op end sten. Da guld er et sjældnere materiale end sten, befinder guldet som for det første i mindre stykker, og for det andet dybere nede. Det betyder at man ofte bliver nødt til at fjerne nogle sten for overhovedet at kunne forfat i alle guldstykkerne.

3.2 Detaljeret beskrivelse af Mineral klassen og inheritance konceptet og brugen

Mineral klassen skal bevirkede som en skabelon til at lave forskellige typer af mineraler udfra. Den skal laves således at man bare kan ændre nogle få variabler i constructoren og derved vil man have et nyt mineral.

Klassen består således af en constructor og metoder til at vise objektet, gemme det specifikke objekts oplysninger i et array, hvis Player kolliderer med det, til brug som argument for Player metoden '**grap()**', og til at ændre en variabel for om mineralet er hejset op.



Figur 1: Flowchart over Mineral klassen og dens underklasser

Klassen bruges som sagt som skabelon for de forskellige mineral typer; Stone og Gold. Det gøres ved brug af OOP konceptet omkring nedarvning; *inheritance*. I generel Java kode skrives;

```

1 class ClassName extends OtherClass {
2     ClassName(type args) {
3         super(args);
4     }
5 }
```

Efter Mineral klassen er skrevet kan man således i en ny klasse bruge inheritance til at nedarve Mineral til den nye klasse. Dette er gjort ved Stone og Gold.

```

1 class Stone extends Mineral {
2
3     //Load sprite image file.
4     PImage r_sprite = loadImage("../sprites/rock.png");
5
6     //Constructor changes a few variables.
7     Stone(int xpos, int ypos, int numberInArray) {
8         super(xpos, ypos, numberInArray);
9         ...
10    }
11 }
```

Listing 1: Stone klassen

```

1 class Gold extends Mineral {
2
3     //Load sprite image file.
4     PImage g_sprite = loadImage("../sprites/gold.png");
5
6     //Constructor changes a few variables.
7     Gold(int xpos, int ypos, int numberInArray) {
8         super(xpos, ypos, numberInArray);
9         ...
10    }
11 }
```

Listing 2: Gold klassen

Hvis man så kigger på metoderne i Mineral klassen, ser man at der er tre;

- display;

```

1     void display() {
2         imageMode(CENTER);
3         image(sprite, x, y, radius, radius);
4     }
```

display bruges som navnet antyder, til at vise objektet på skærmen. Dette gøres ved brug af Processings funktion, *image()*, der tager en bildefil, et koordinatsæt og en størrelse, som argument. Koordinatsættet er placering på objektet og størrelsen sættes som radius på objektet.

Man kan se at *display* ikke returner noget, da det har datatypen void, hvilket betyder ”manglen på noget”, og den tager ikke mod nogen argumenter.

- mineralCollision;

```
1 int [] mineralCollision(float a, float b) {  
2     int [] mineralCollision = new int[2];  
3     if (dist(a, b, x, y) < radius) {  
4         mineralCollision[0] = n;  
5         mineralCollision[1] = type;  
6     }  
7     return mineralCollision;  
8 }
```

mineralCollision bruges til at gemme de nødvendige oplysninger om et specifikt objekt til brug i andre funktioner i programmet. Dermed er den sat op til at have en datatype som integer (int). Men for ikke at skulle lave flere funktioner der returnere en integer, har jeg valgt at mineralCollision skal returnere et array af integers.

- hasCaught;

```
1 boolean hasCaught() {  
2     return caught = true;  
3 }
```

hasCaught ændrer en variabel for et specifikt Mineral objekt. Variablen der skal ændres er af datatypen boolean, derved skal metoden også returnere en boolean.

3.3 Funktion og metode oversigt for GoldMiner

class	name	type	args	usecase
Main	settings	void		Bruges til at sætte indstillingerne for programmet. Heri vinduestørrelse med <i>size()</i> . Køres en enkelt gang.
Main	setup	void		Bygger videre på <i>settings()</i> ved at sætte frameraten, loade baggrundsbille, sætte setupphase loopet og tilføje alle Mineral objekterne til ArrayList. Køres en enkelt gang.
Main	draw	void		Funktion der styrer hovedprogram loopet. <i>draw</i> køres hvert frame.
Main	regen	void		Erstatter Mineral objekter i ArrayList med nye.
Main	overlap	boolean	float, float, float, float, float, float	Tjekker om der er overlap mellem input punkterne. Returner sand hvis og false hvis ikke.

Main	keyPressed	void	Sender tastatur input om en knap er trykket ned til <i>Player.setMove</i> .
Main	keyReleased	void	Sender tastatur input om en knap er sluppet til <i>Player.setMove</i> .
Mineral	display	void	Metoden rendere objektet ved <i>image()</i>
Mineral	mineralCollision	int array float, float	Metoden laver et array på 2. Hvis distancen mellem input argumenterne og Mineral objektets <i>x</i> og <i>y</i> variabel (koordinater), så sættes [0] = <i>n</i> og [1] = <i>type</i> , hvilket betyder at oplysningerne for hvilket Mineral objekt, der er kollideret med input argumenterne, returneres.
Mineral	hasCaught	boolean	Metoden ændrer variablen, <i>caught</i> til <i>true</i> , og dermed bliver objektet tagget til at fjernes.

Player	display	void		Del 1 af rendering af objekt. Metoden bruges til at vise objektet ved <i>circle()</i> .
Player	update	void		Del 2 af rendering af objekt. Metoden bruges til at opdatere variablerne som bruges i <i>display()</i> .
Player	grap	void	int, int	Metode til at køre metoder og funktioner, når en kollision mellem Player objekt og et Mineral objekt finder sted. Metoderne der køres; <i>Score.calcMoney</i> , <i>Mine-ral.hasCaught()</i> og <i>pReset()</i> . Samt manipulere med Player objekt, for at trække tilbage automatisk og genstarte <i>update()</i> .
Player	setMove	boolean	char, boolean	Et ”switch-case” scenarie. Metode til at oversætte tastatur inputs til boolean variabler.

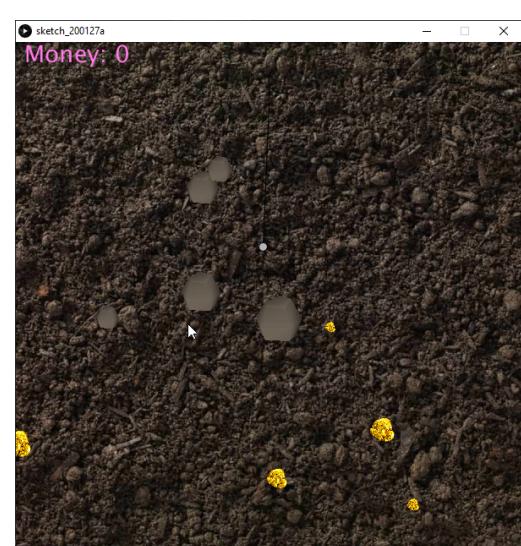
Player	movement	void	Metode til at oversætte boolean variabler fra <i>setMove()</i> til objekt variabler og køre <i>pReset()</i> .
Player	pReset	void	Metode til at sætte Player objektet til start værdier.
Score	display	void	Metoden rendere objektet ved <i>text()</i> .
Score	calcMoney	int	Metode til at beregne hvor meget der skal tilføjes til <i>money</i> baseret på et specifikt Mineral objekt. Returnerer <i>moneyAdd</i> .

4 Test af spillet

Til at test programmet compiler jeg det og prøver at gennemføre spillet.



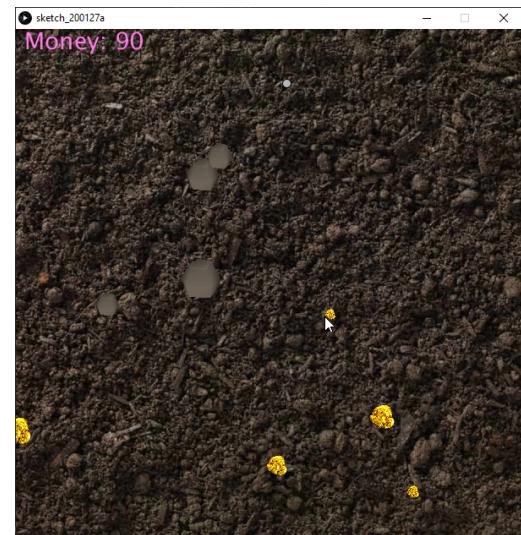
(a) Opstart af spillet, med generede mineraler. Player armen svinger frem og tilbage



(b) Brugeren trykker s og armen rækkes ud

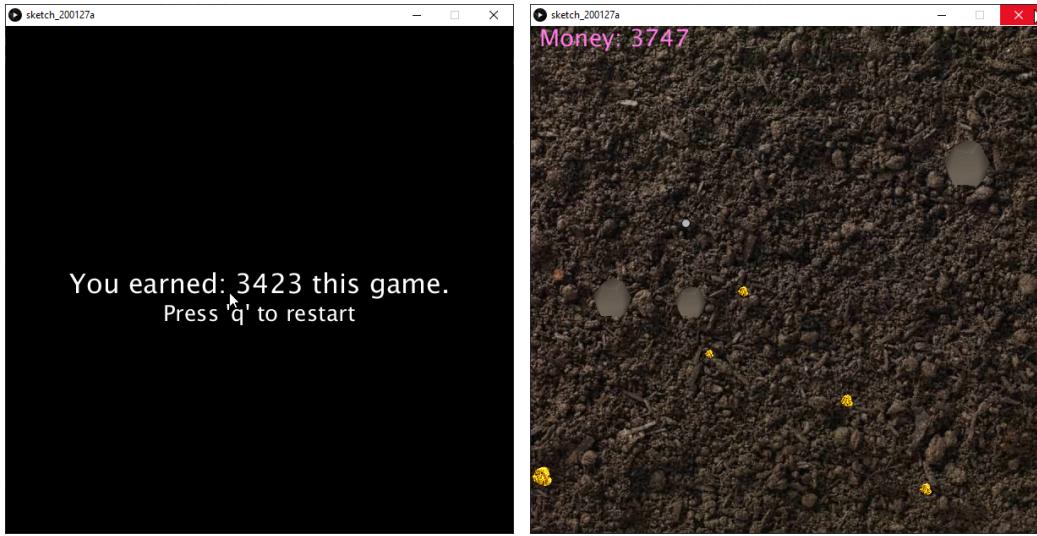


(c) Armen har ramt et mineral og trækkes tilbage.



(d) Penge bliver tilføjet og armen begynder at svinge igen.

Figur 2: Gameplay af GoldMiner



(a) Når man har samlet alle mineralerne vises slutskærmen med ens score.
 (b) Brugeren har trykket *q* og et nyt level starter.

Figur 3: Slutskærm og nybane.

Forbedring / viderebygningsmuligheder

- **Opgraderinger;** ind i mellem levels kunne man lave et opgraderings-system, så man kunne få mulighed for at gøre sin opsamler større så det er lettere at ramme.
- **Flere typer af mineraler;** man kunne introducere flere slags mineraler.
- **Bedre mineral generation;** optimering af valg af placering til mineralerne.
- **Start menu;** en start menu med Spil, How to play og highscores kunne introduceres for at gøre det mere brugervenligt.
- **Sprite til Player;** istedet for en sort streg og lysegrå kugle kunne man give Player en ordentlig sprite ligesom mineralerne.
- **Highscore system;** man kunne implementere et highscore system, der kan gemme ens score og kan man se den i en highscore-menu.

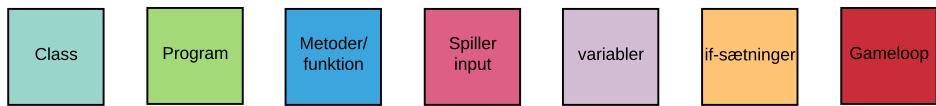
5 Konklusion

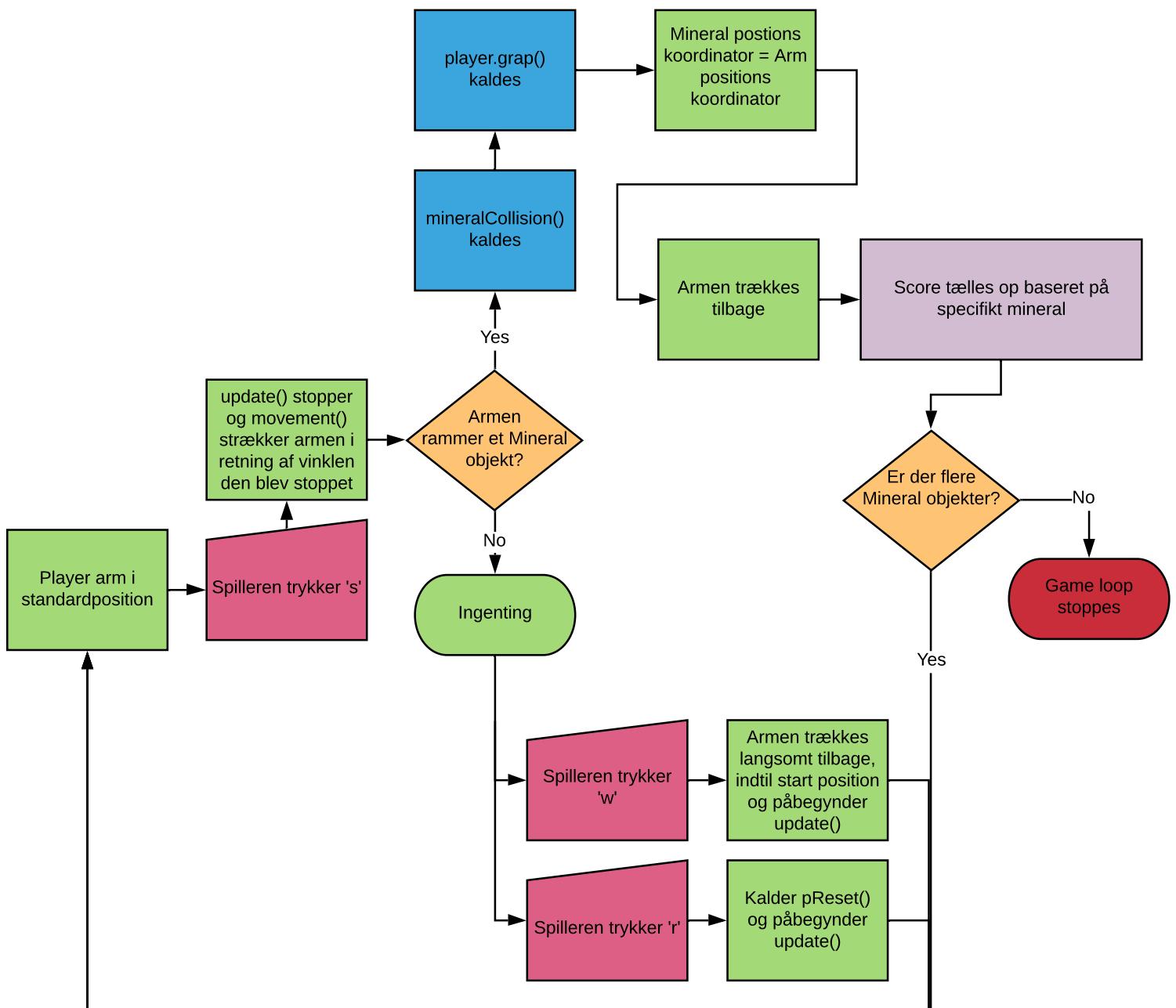
For at konkludere på projektet er der blevet lavet et program, som efter tidsrammen ligner det originale Gold Miner forholdsvis godt. Programmet har stadigvæk en del mangler og kunne bruge en del optimering.

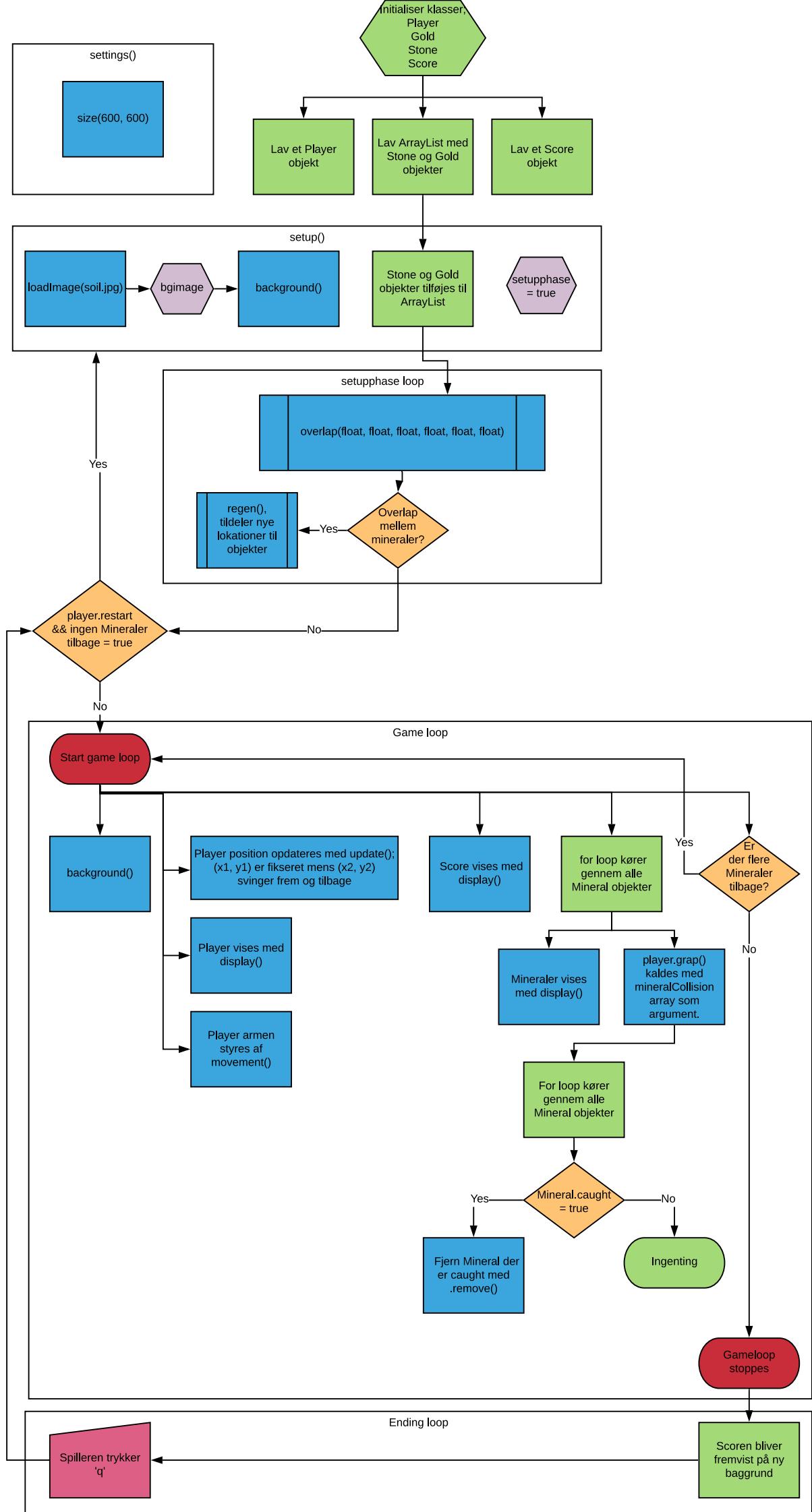
Spillet har haft indflydelse fra matematik om hvordan punkter beregnes ud fra andre punkter ved hjælp af de trigonometriske funktioner.

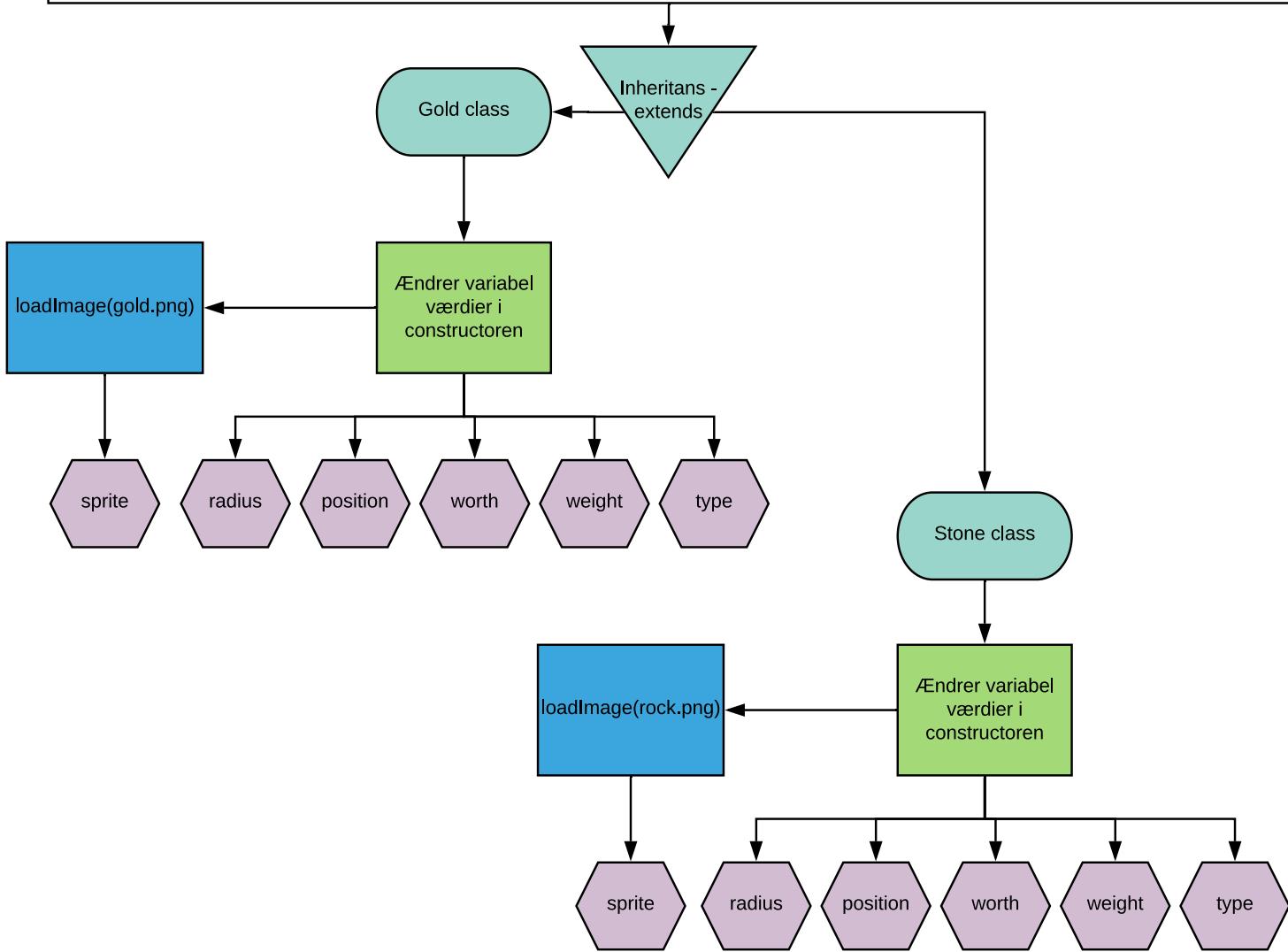
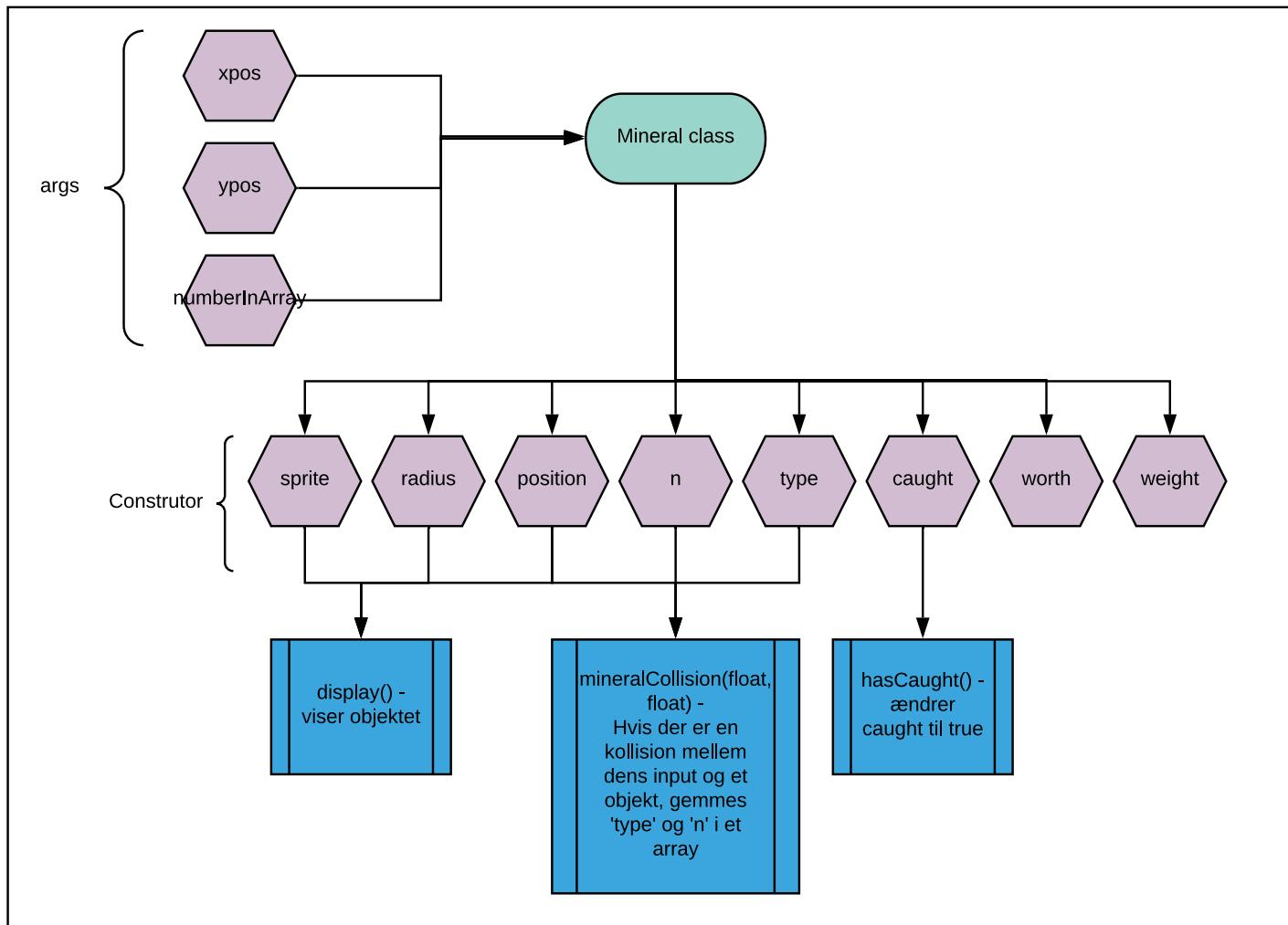
6 Bilag - Flowcharts og source kode

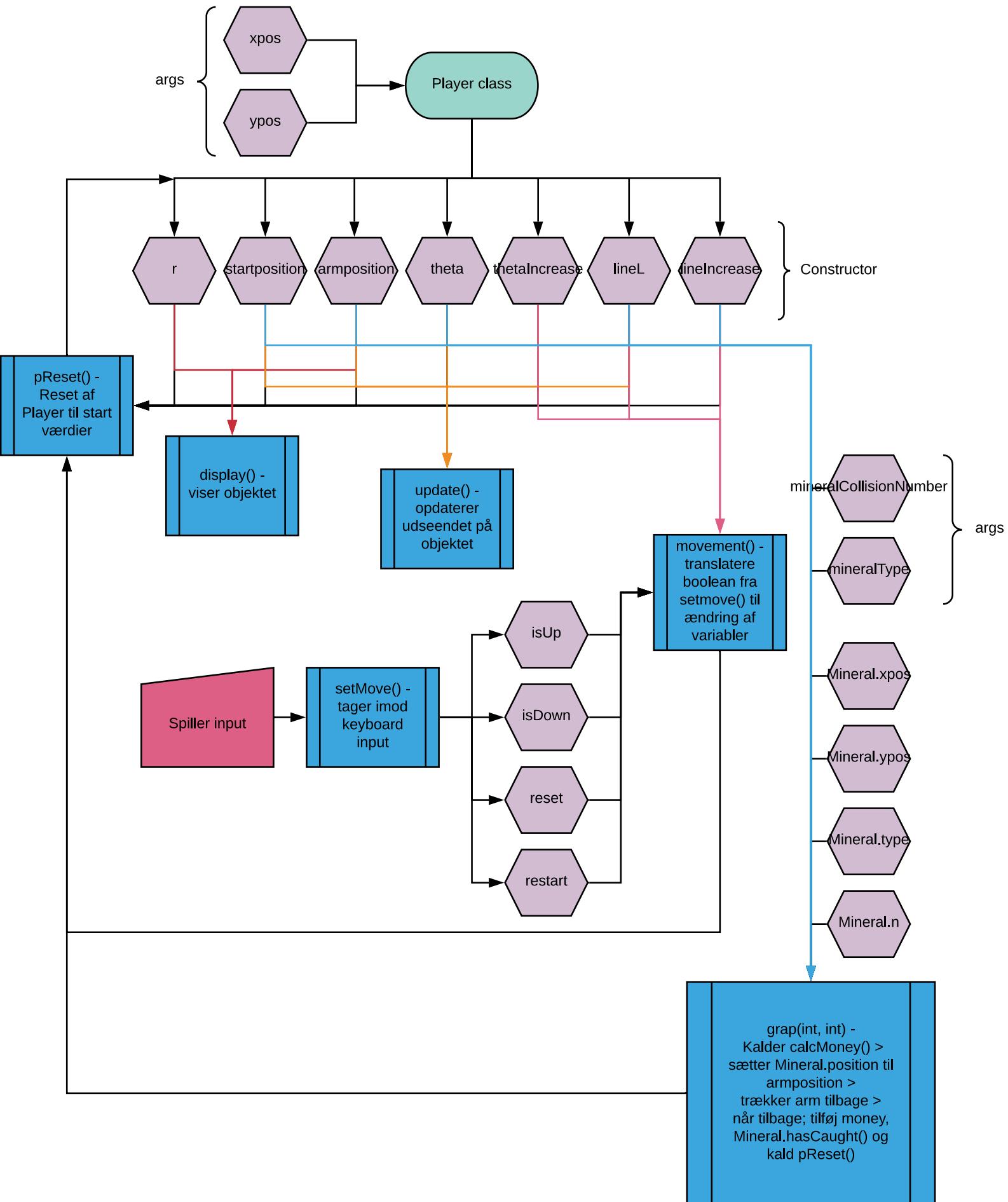
Forklaring

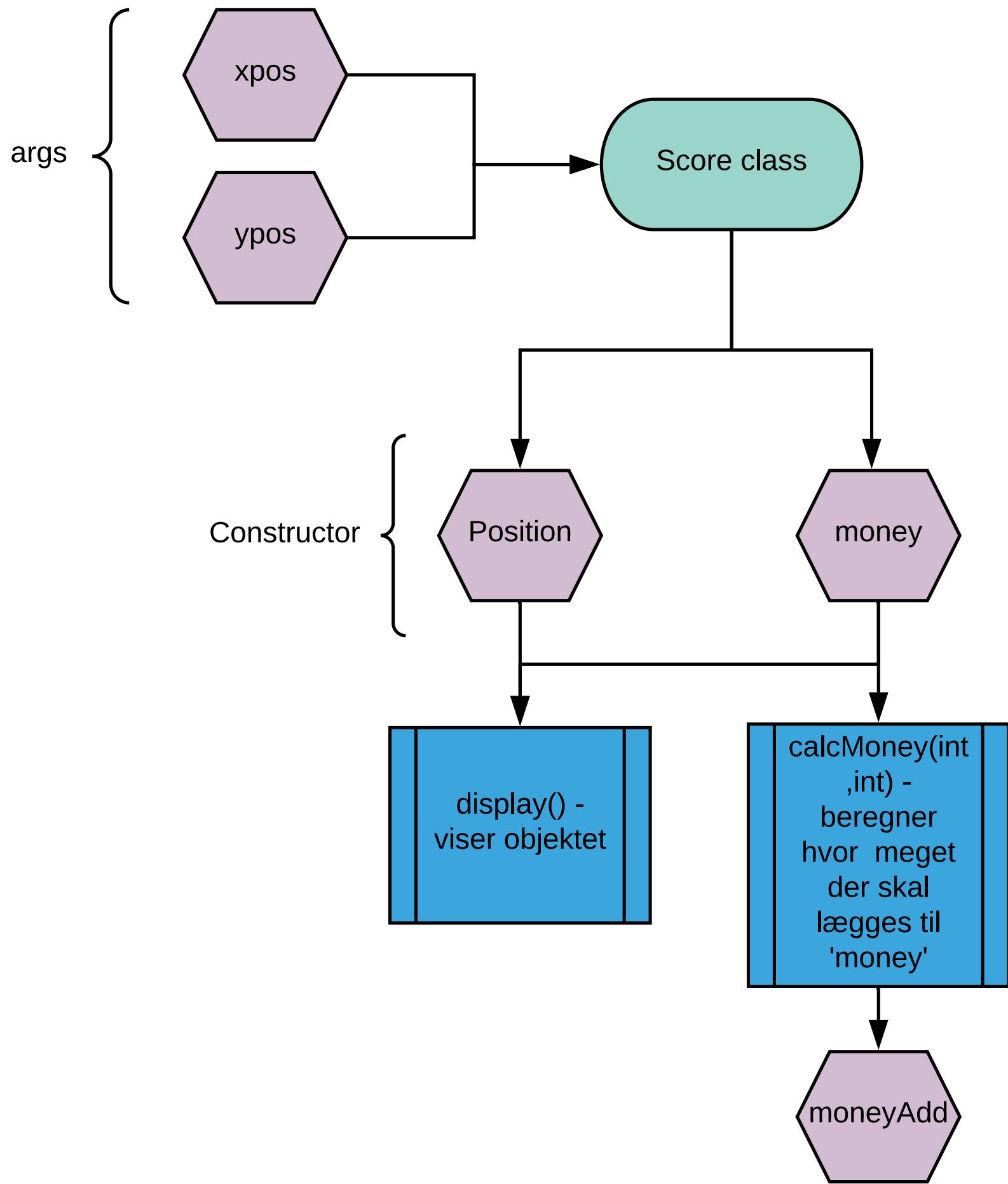












sketch_200127a

//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 'q' > 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]); //mineralCollisionNumber[0] = number in array, mineralCollisionNumber[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]); //mineralCollisionNumber[0] = number in array, mineralCollisionNumber[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 'q' to restart", width/2, height/2+36);
}
}

//Function for replacing Stone and Gold objects. 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'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 graph(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 'true' 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 'true' 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 'w':
            return isUp = b;
        case 's':
            return isDown = b;
        case 'r':
            return reset = b;
        case 'q':
            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;  
}
```