Bomberman

## Einleitung

Bomberman ist üblicherweise ein Multiplayerspiel bei dem ein Spieler zu Beginn auf einer Karte an unterschiedlichen Punkten gleichzeitig startet.

Ziel ist es alle anderen Spieler mittels Bomben, die man platzieren kann auszulöschen.


*(Abbildung 1: Bomberman Screenshot. Quelle: https://www.maniac.de/wp-content/uploads/2017/03/08-14-640.jpg)*

Ich möchte eine ziemlich ähnliche Abbildung des Spiels Programmieren. (Begrenzt auf 3 Items)

Das bedeutet:

- Multiplayer (in meinem Fall erstmal ein 2 Spieler System)
- Jeder Spieler kann mittels Tastendruckes eine Bombe platzieren. Bevor diese nicht hochgegangen ist, kann er keine weitere platzieren.
- Der Explosionsradius einer Bombe ist genau ein Feld einer Mauer in jede Achse.
- Karte auf 15x15 Blöcke aufgebaut:
    - Unzerstörbare Wände
    - Zerstörbare Wände, dahinter kann sich ein Item verbergen

  Items können folgendes bewirken:

- Ein Explosionsverstärker = Diese erhöht den Explosionsradius um ein weiteres Feld in jede Achse. (Häufigkeit = Gewöhnlich)
- Ein Bombenslot = dieser lässt eine weitere Bombe zeitgleich zu dem anderen Platzieren. (Häufigkeit = Gewöhnlich)
- Ein Rollschuh = mit diesem kann man schneller laufen (Häufigkeit = Sehr selten)

## Spieler

Jeder Spieler besitzt das gleiche Aussehen, jedoch eine andere Farbe.

Spieler 1 – wird links oben in der Karte starten – dieser kann mit den Tasten ‚W A S D' sich bewegen und mit der ‚LEERTASTE' eine Bombe platzieren.

Spieler 2 – wird recht unten in der Karte starten – dieser kann sich mit den ‚PFEILTASTEN' bewegen und mit der ‚ALT GR' eine Bombe platzieren.

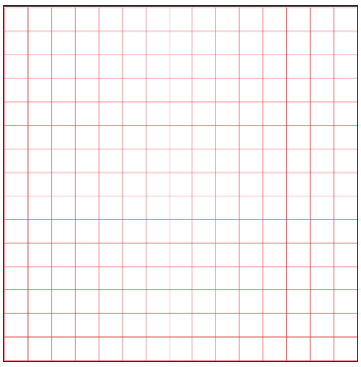
Jeder Spieler hat 3 Leben.

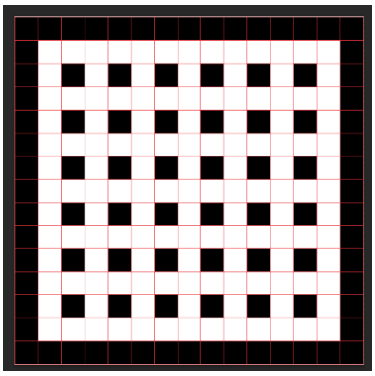
## Bomben

Die Bomben werden sofort nach dem Drücken der Taste auf der Position des platzierenden Spielers platziert. Diese Bombe geht nach ca. 2 Sekunden hoch.

## Karte

Die Karte soll gleichmäßig verteilte „nicht-zerstörbare" Wände haben.





Die Blockmenge ist = 15 Blöcke x 15 Blöcke | a 50px. Sind also 750px x 750px.

```
class Karte {


  int [][] map = {
    {0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,0},
    {0,1,1,1,2,  2,1,2,1,2,  2,2,2,2,0},
    {0,1,0,1,0,  2,0,2,0,2,  0,2,0,2,0},
    {0,1,1,1,2,  2,1,2,1,2,  2,2,2,2,0},
    {0,2,0,2,0,  2,0,2,0,2,  0,2,0,2,0},

    {0,2,2,2,2,  2,1,2,1,2,  2,2,2,2,0},
    {0,1,0,1,0,  1,0,2,0,1,  0,1,0,1,0},
    {0,2,2,2,2,  2,2,2,2,2,  2,2,2,2,0},
    {0,1,0,1,0,  1,0,2,0,1,  0,1,0,1,0},
    {0,2,2,2,2,  2,1,2,1,2,  2,2,2,2,0},

    {0,2,0,2,0,  2,0,2,0,2,  0,2,0,2,0},
    {0,2,2,2,2,  2,1,2,1,2,  2,1,1,1,0},
    {0,2,0,2,0,  2,0,2,0,2,  0,1,0,1,0},
    {0,2,2,2,2,  2,1,2,1,2,  2,1,1,1,0},
    {0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,0}
};
```

```
imageMode(CORNER);
for (int i = 0; i < map.length; i++) {
  for (int j = 0; j < map[i].length; j++) {
    if (map[i][j] == 0) {
      image (wall0, j * block, i * block, block, block);
    } else if (map[i][j] == 1) {
      image (wall1, j * block, i * block, block, block);
    } else if (map[i][j] == 2) {
      image (wall2, j * block, i * block, block, block);
    } else if (map [i][j] == 3) {
      image (wall1, j * block, i * block, block, block);
    }
    //rect(j * block, i * block, block, block);
  }
}
```

 = wall0  = wall1  =wall2

## Items

Da ich Items ins Spiel einbringen möchte, habe ich viele Variablen verwendet.

0 = Feld ohne Item

1 = Bomben Item

2 = Explosion Item

3 = Speed Item

Diese Items werden auf mit einem Zufallsgenerator an derselben Stelle wie :

MAP [x][y] == 4

erzeugt.

Eine 4 auf MAP[x][y] entsteht, wenn, eine Explosion mit einer zerstörbaren Wand kollidiert.

## Bomben

Die Bomben werden mit einer ARRAYLIST angelegt. Die maximale länge darf die maximale Bombenanzahl im Inventar nicht überschreiten, außer, diese in der „Explosionphase“.

Hierfür habe ich einen extra Zähler verwendet.

Sobald die Explosion vorbei ist, wird die Bombe gelöscht.

## Explosionen

Die Explosion ist nur maximal so lange wie auch die Variable explosionRadius ist, es sei denn:

- Es befindet sich eine „hardwall“ oder eine „destroyable wall“ davor.

Die Kollision der Explosionen überprüfe ich wiederrum, indem ich für jede Explosion ein 2D Byte Array anlege. Dafür habe ich die Daten der normalen Map genommen, die Explosion darübergelegt und mit dieser dann entsprechend weitergearbeitet.

## Keybinds

Die Keybinds übergebe ich über einen Array in die Methode in der daraus ein Boolean initialisiert wird.

**Wichtig: Der Spieler soll nicht quer laufen können.**

Deshalb wird immer der zuletzt gedrückte Knopf ausgeführt.

## Collisiondetection

Um die Kollision zwischen dem Spieler und der Wand zu prüfen habe ich geschaut ob der Spieler sich in Feld 1 befindet, dort darf er sich bewegen.

Diese Annahme ist jedoch falsch, da wenn er in Feld 2 rutsch nicht mehr rauskommt.

Also habe ich eine Methode geschrieben, die prüft, ob AKTUELLE POSTION + SPEED eine Kollision mit der Wand wird.

Falls ja ist der schritt nicht möglich und der Spieler unternimmt nichts.

```
if (keys[0]) {
  if (map [int(((pos.y-speed)-(pHeight/2))/block)][int((pos.x+(pWidth/2))/block)] == 1 &&
      map [int(((pos.y-speed)-(pHeight/2))/block)][int((pos.x-(pWidth/2))/block)] == 1 ){
    pos.y -= speed;
    imageMode (CENTER);
    PImage currentPic = playerPic.get(24 + (24 * ((frameCount/10)%2)),64,24,32);
    image (currentPic, pos.x, pos.y - correcture,imageFactor * pWidth, imageFactor * pHeight);
    lastDirection = 2;
  } else {
     imageMode (CENTER);
    PImage currentPic = playerPic.get(24 + (24 * ((frameCount/10)%2)),64,24,32);
    image (currentPic, pos.x, pos.y - correcture, imageFactor * pWidth, imageFactor * pHeight);
    lastDirection = 2;
  }
}
```

*BEISPIEL FÜR DIE W TASTE*

**Zugleich soll er nicht durch eine Bombe laufen können.** Da der Spieler zum Zeitpunkt des Platzierens der Bombe, sich in der befindet, muss er jedoch erstmal rauskönnen.

Dieses Problem habe ich mit einem einfachen Boolean gelöst:

```
void checkBomb (PVector bombPos, boolean bombActive) {
  if ( ( ((pos.x) > (bombPos.x - 40) && (pos.x) < (bombPos.x + 40) )  &&  ( (pos.y) > (bombPos.y - 40) && (pos.y) < (bombPos.y + 40) ) )   ) {
  playerInBomb = true;

  } else {
  playerInBomb = false;

  }
```

Ob der Spieler ein Item aufgesammelt hat, prüfe ich, indem ich eine Map für die Items anlege und überprüfe, ob die Position des Spieler mit der des Item übereinstimmt.

Dieses mache ich wieder mit einem 2D Byte Array.

```
void change (int [][]  explosionMap, boolean explosionActive, boolean bombActive) {
  for (int i = 0; i < explosionMap.length; i++) {
    for (int j = 0; j < explosionMap[i].length; j++) {
      if (explosionMap [i][j] == 3 && map[i][j] == 2) {

        if (explosionActive && bombActive) {
       map [i][j] = 1;


       int item = int (random (1,50));

       // 1: Bomb+ Item | 2: Radius+ Item | 3: Speed+ Item
       if (item >= 1 && item <= 12) {
         itemMap [i][j] = 1;
         } else if (item > 12 && item <= 20) {
           itemMap [i][j] = 2;
         } else if (item > 20 && item <= 23) {
           itemMap [i][j] = 3;
         }

        }


      }
    }
  }
}
```

Bombhit

Befindet sich der Spieler in eines der **explosionMap [x][y]=1** Feld, kriegt der ein Leben abgezogen, kann sich in dem Zeitpunkt nicht bewegen, es wird eine Animation abgespielt und daraufhin wird die X und Y Position zurückgesetzt. Währen dieser Zeit ist der Spieler unverwundbar + 2 Sekunden nach dem er neu „spawnt“ auch. Ein sogenannter Cooldown

## Sprites

Da ich den Code schon entsprechend aufgebaut habe musste ich nur an die entsprechenden stellen des Sprites einsetzten und anwenden.

Ich musste nur für die Drehung des Spielers, eine extra Methode und eine Variable einrichten.

Beispiel: Spieler steht und schaut nach recht, dann läuft er nach unten und bleibt anschließend stehen. NUN SOLL DER SPIELR AUCH NACH UNTEN SCHAUEN.

Ich speicherte hierfür die **int lastDirection** über die Keys, die ich drücke.

## GameModes

Gamemode == 0 : Das Startmenü, hier kann der Spieler das Spiel schließen, zum Gamemode == 1 -> Einstellungen, oder zum Gamemode == 2 -> Das eigentliche Spiel.

Gamemode == 2 : Das Spiel soll nicht plötzlich anfangen, sondern es soll über ein Timer runterlaufen. Sobald ein Spieler oder beide 0 leben haben -> Gamemode = 3.

Gamemode == 3 : Hier wird der Gewinner mit einer Animation angezeigt. Danach kann er per Tastendruck wieder zu Gamemode = 0 wechseln.