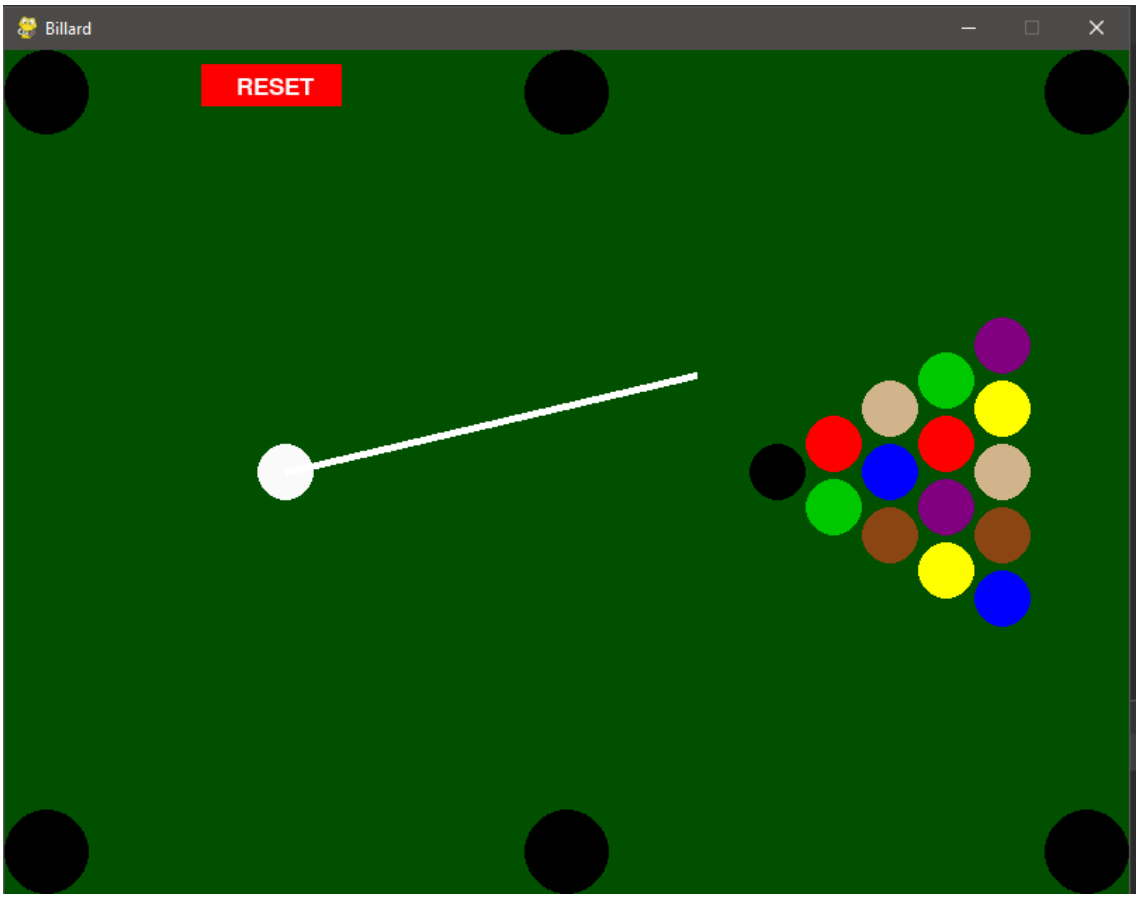
Projekttitel: Ein Billard Spiel

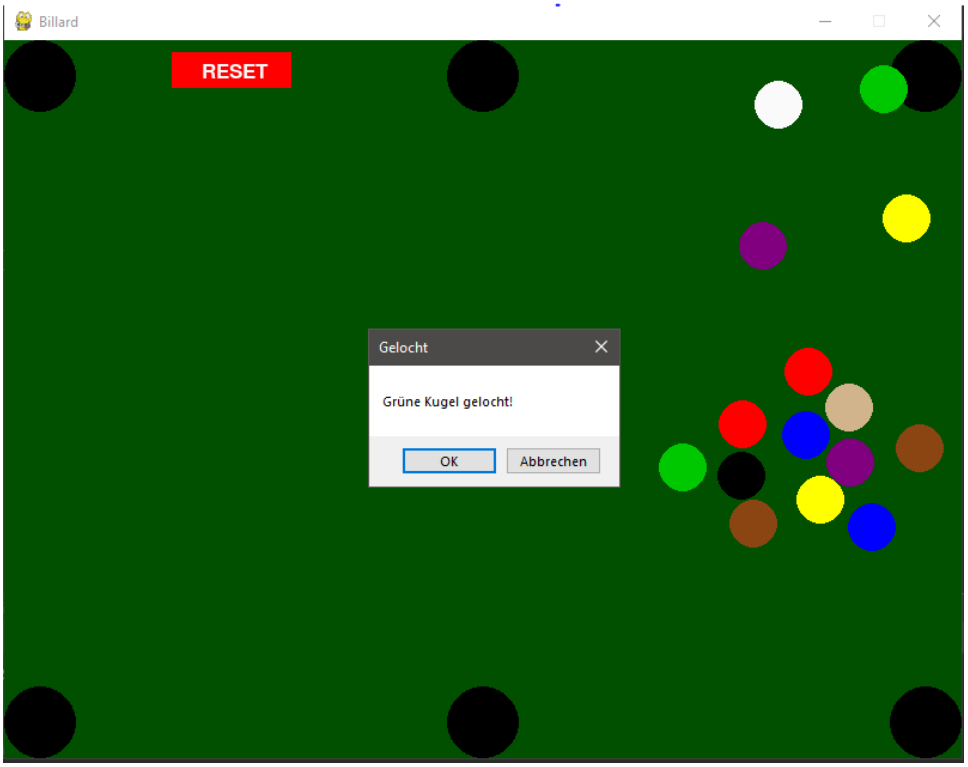
Teilnehmer: Mark Kostmann

Allgemeine Informationen:

- Es wird ein grüner Billardtisch, auf dem sich 6 Taschen am Rand des Spielfeldes befinden dargestellt.
- Im mittleren Bereich des linken Spielfeldes wird eine weiße Kugel mit einem Queue angezeigt.
- Im mittleren Bereich des rechten Spielfeldes werden 15 farbige Kugeln in einem Dreieck angeordnet. Hierbei gibt es von jeder Farbe zwei Kugeln. Aber es gibt nur eine schwarze Kugel.



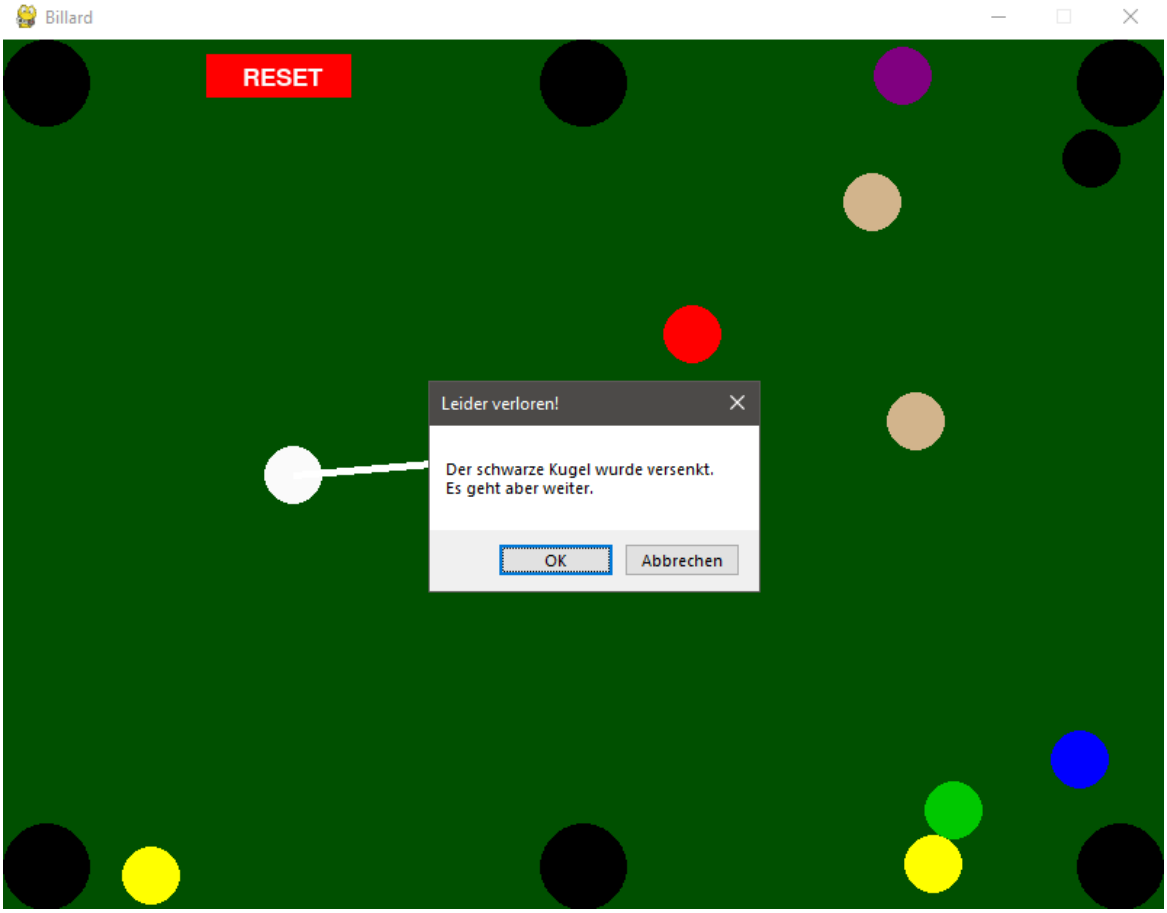
- Ziel ist es, die farbigen Kugeln mit der weißen Kugel in die Taschen zu spielen. Wobei die schwarze Kugel als letzte Kugel eingespielt werden soll.
- Wenn eine farbige Kugel eingespielt wurde erscheint eine Msgbox, die besagt welche Kugelfarbe eingespielt wurde.



- Wenn die weiße Kugel in eine Tasche gespielt wurde wird sie wieder in die Ausgangsposition gesetzt. Falls sich dort an der Position bereits eine andere Kugel befindet überlappen sich die Kugeln. Eine entsprechende Korrektur war mir in Anbetracht der Zeit nicht gelungen.

- Wenn die schwarze Kugel vor den anderen farbigen Kugeln eingespielt wurde hat man das Spiel verloren. Es wird eine Msgbox angezeigt, dass man das Spiel verloren hat.

  Dennoch geht das Spiel weiter und die schwarze Kugel wird im rechten Spielfeld positioniert. Falls sich dort an der Position bereits eine andere Kugel befindet überlappen sich die schwarze Kugel mit der dort vorhandenen Kugel. Eine entsprechende Korrektur war mir in Anbetracht der Zeit nicht gelungen.



- Wenn die schwarze Kugel als letzte Kugel eingespielt wurde erscheint eine Msgbox, die sagt, dass man das Spiel gewonnen hat.
- Es gibt einen RESET Button mit dem man das Spiel wieder auf den Startzustand bringen kann.

Wie das Programm bedient wird:

Beim Start sieht man auf dem linken Bereich eine weiße Kugel und im rechten Bereich befinden sich 15 farbige Kugel. Je nachdem wie weit man sich mit dem Maus-Cursor von der weißen Kugel entfernt befindet wird ein weißes Queue an der weißen Kugel angezeigt. Die maximale Länge die ein Queue einnehmen kann sind 300 Pixel. Je länger das Queue desto heftiger wird der Impuls sein, wenn man das Queue anklickt. Die Länge des Queues kann man durch Verschieben des Maus Cursors beeinflussen.

Das Einspielen der weißen Kugel bewirkt, dass die weiße Kugel wieder an der Startposition gesetzt wird.

Wenn die schwarze Kugel als letzte Kugel eingespielt wird … siehe oben

Wenn man den RESET Button klickt wird das Spiel wieder von vorne gestartet und die Ausgangsposition der Kugeln erscheint wieder.

Aufbau des Projekts

Wie wurde dieses Projekt aufgebaut?

Das Billard Spiel verwendet drei Klassen:

1. Class Kugel
    a. Im Konstruktor einer Kugel
    werden die x- und y- Koordinaten gespeichert ebenso die Kugelfarbe.
    b. **drawBall(self, window)**
    In dieser Methode wird mit dem importierten Package und der dortigen Methode eine Kreiskugel gezeichnet
    c. move(self)
    Bewegt eine Kugel Anhand der neuen x- und y- Koordinaten
    d. **checkWallCollision(self, window_width, window_height)**
    Hier wird die Geschwindigkeit von vel_x und vel_y mit dem Faktor -1 multipliziert, so dass sich die Richtung umdereht.
    e. **setPosition(self, x, y)**
    Die Position einer Kugel wird anhand der x- und y- Koordinate festgelegt.
    f. **isOverlapping(self, other_ball)**
    Hier wird die Kollision der Kugeln berechnet. Das einfachste Gesetzt dabei ist der Pytagoras. Ansonsten wird hier mit den physikalischen Gesetzen gearbeitet.
2. Class Pocket
    a. Im Konstruktor eines Pockets
    werden die x- und y- Koordinaten, die Farbe und der Pocket-Radius eingetragen.
    b. **drawPocket(self, window)**
    In dieser Methode wird mit dem importierten Package und der dortigen Methode eine Kreisfläche gezeichnet

3. Class Queue
    a. Im Konstruktor eines Queues
    werden die x- und y- Koordinaten, die Anfangslänge des Queues und die Sichtbarkeit auf True eingetragen.
    b. **createQueuet(self)**
    Erzeugt das Queue und stellt die maximale Länge des Queues auf 300 ein
    c. **drawQueue(self, window)**
    Zeichnet das Queue

Genauere Beschreibung der Methoden weiter unten

Dateien

Das Spiel wird mit der Datei **billard.py** aufgerufen. Im Footer der Funktion habe ich eine Abfrage erstellt, dass das Spiel nur funktioniert, wenn es direkt aus der Datei aufgerufen wird. Man kann die Spieledatei also nicht einbinden und so das Billard Spiel spielen. Es wurden verschiedene Dateien angelegt. Jede Klasse befindet sich in einer extra Klassendatei. Ferner wurden die Funktionen ausgelagert und befinden sich in der **funky.py**.

Dateien:

- **billard.py**
  In dieser Datei wird das Spiel arrangiert. In dieser Datei werden folgende Pakete bzw. Klassen importiert:
    - import pygame
    - import sys
    - from Queue import Queue
    - from Pocket import Pocket
    - from funky import)

  In dieser Datei wird das Billard Spiel gesteuert.

  Zunächst wird das importierte Packet pygame mit pygame.init() initialisiert. Ein leeres Array (Liste) wird erstellt. Anschließend werden die Bildschirmabmessungen angegeben und in einer public Variable gespeichert. Danach werden Farben namentlich definiert und festgelegt.

  Die Funktion „main“ wird dann aufgebaut und in dieser Funktion findet das Spiel statt. Die Funktion wird ganz unten im Skript aufgerufen.

    - **Funktion main:**
      Zunächst wird das Spielfeld aufgebaut. Zuerst werden die vorher definierten Farben in ein Array (Liste) gespeichert. Anschließend werden die Kugeln Positions mäßig festgelegt. Diese Kugeldaten werden in einem Array (Liste) gespeichert. Danach wird das Array mit den Farben und das Array mit den Positionen in ein weiteres Array (Liste) gespeichert. Hier haben wir nun einen Array in ein Array.

      Ein Queue wird erzeugt in dem die Klasse Queue aufgerufen wird. Hier werden die x- und y- Koordinaten der weißen Kugel übergeben: Queue(balls[0].x, balls[0].y)

      Danach werden die 6 Taschen erzeugt indem 6 mal die Klasse Pocket zum initialisieren der 6 Objekte Taschen aufgerufen wird. Die 6 Pocket-Objekte werden in einem Array pockets gespeichert.

      Danach wird der RESET Button erzeugt. Zuletzt wird das Queue auf Sichtbar

gestellt und ein Dämpfungsfaktor festgelegt. Der Dämpfungsfaktor bewirkt, dass das Rollen der Kugeln im Laufe der Zeit immer langsamer wird.

Damit ist das Spielfeld aufgebaut.

Das Spiel befindet sich in einer while Schleife, die quasi endlos läuft. Es gibt eine Variable, playing genannt, die auf True gesetzt ist. Wenn playing auf False gesetzt wird, wird die Schleife beendet.

Zunächst wird entsprechend des Maus-Cursors das Queue an die weiße Kugel gebracht. Es wird zuerst geprüft ob das Queue benutzt wurde und wenn ja, dann wird entsprechend des Impulses der weißen Kugel die Bewegungsrichtung und die Geschwindigkeit der angestoßenen Kugel berechnet und eingestellt.
Des Weiteren finden diverse Prüfungen statt. Auch werden (solange das Queue nicht sichtbar ist) die Positionen und Geschwindigkeiten der anderen angestoßenen Kugeln berechnet. Gleichzeitig werden die Bewegungen der Kugeln überwacht und wenn ein Schwellwert von <= 0.1 erreicht wurde wird die entsprechende Kugel gestoppt. Auch werden die Kollisionen der Kugeln untereinander überwacht und die Kugeln bewegen sich entsprechend des Impulses in eine andere Richtung.

Zu Schluss wird überprüft ob eine (oder mehrere) Kugel in eine Tasche eingespielt wurde.

- **funky.py**
  In dieser Datei wurden **die Funktionen ausgelagert.** In dieser Funktions-Datei finden folgende importe statt:
  - import math
  - import ctypes
  - from Kugel import Kugel
  - # Import um die Farbkonstanten einzubeziehen
  - from billard import RED, GREEN, BLUE, BROWN, LIGHT_BROWN, YELLOW, VIOLET, BLACK

  Die Funktionen werden während des Spielens die ganze Zeit aufgerufen; somit kann man auch zum Beispiel prüfen, ob eine Kugel in eine Tasche eingespielt wurde.

  Es gibt folgende Funktionen:

  - **resetGame(white_ball, colored_balls, ball_colors, positions, WINDOW_WIDTH, WINDOW_HEIGHT)**
    Damit wird das Spiel wieder auf den Start gebracht. Die notwenigen Daten werden als Parameter mit übergeben.

Zunächst wird abgefragt, ob sich die weiße Kugel unter den Farbigen Kugeln befindet. Wenn ja, dann wird die weiße Kugel entfernt und gleich wieder wird die weiße Kugel mit den x- und y- Koordinaten auf den Wert 0 (Null) gesetzt. Zum Schluss werden die farbigen Kugeln, wie zuvor im Parameter „colored_balls“ definiert, als Dreieck im rechten Spielfeld arrangiert.

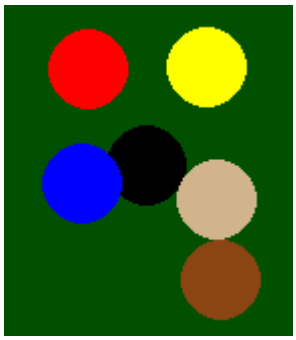
- **checkPocketCollision(ball, pockets)**
  Hier wird der Abstand einer Kugel zum Fangkorb ermittelt. Ist die ermittelte Distanz <= dem Radius der Tasche dann gilt die Kugel als eingespielt.

- **arrangeBallsAtPositions**
  Hier werden lediglich die farbigen Kugeln entsprechend den vorher festgelegten x- und y- Werten positioniert. In unserem Fall werden die Kugeln in einem Dreieck in der rechten Spielfeldbereich angeordnet.

- **msgbox(title, text)**
  Die Funktion bekommt einen Titel und einen Text. In der Datei wurde die „**ctypes**“ importieret und damit kann man ein Ausgabe-Popup erzeugen. Der Popup zeigt den Title im oberen Rand und den Text im unteren Bereich an.

- **gameStatus(ball)**
  Wenn eine Kugel eingelocht wurde wird diese Funktion aufgerufen. Sie prüft nur ob es sich nicht um die weiße Kugel handelt und gibt lediglich die Farbe der eingespielten Kugel zurück. Diese zurückgegebene Farbe wird von der Hauptfunktion, die diese Funktion aufgerufen hat, in ein Array (Liste) gespeichert.

- **setBlackBall(WINDOW_WIDTH, WINDOW_HEIGHT, ball)**
  Setzt die schwarze Kugel wieder zurück ins Spiel. Eine vernünftige Kollisionsbearbeitung war mir da nicht gelungen und so kann es vorkommen, dass die schwarze Kugel von anderen Kugeln überlagert wird.



- **setWhiteBall(WINDOW_WIDTH, WINDOW_HEIGHT, ball, queue)**
  Setzt die weiße Kugel wieder zurück ins Spiel. Eine vernünftige Kollisionsbearbeitung war mir da auch nicht gelungen und so kann es

vorkommen, dass die weiße Kugel von anderen Kugeln überlagert wird.

- **Klasse in Kugel.py**
  Diese Klasse erzeugt Kugeln und prüft eine Kollision mit anderen Kugeln. Die Kollision enthält leider Fehler und die Fehler waren mir in Anbetracht der Zeit nicht gelungen zu eliminieren.
  In dieser Klasse finden folgende Importe statt:

  - **import pygame**
  - **import math**

  Folgende Methoden gibt es in dieser Klasse:

  - **__init__(self, x, y, color)**
    Im Konstruktor werden die x- und y- Position gesetzt. Ebenso wird die Farbe, der Radius default auf 20, die Sichtbarkeit und die Geschwindigkeit gesetzt. Die Geschwindigkeit wird default mäßig auf 0 (Null) gesetzt.

  - **drawBall(self, window)**
    Wenn eine Kugel die Eigenschaft Sichtbar auf True gesetzt hat, dann wird mit dem pygame Paket ein Kreis mit einem Radius und einer Farbe gezeichnet. Diese Kugel wird an die entsprechende x- und y- Position platziert.

  - **move(self)**
    Setzt die Kugel entsprechend der Geschwindigkeit auf eine neu x- und y- Position.

  - **checkWallCollision(self, window_width, window_height)**
    Prüft, ob die Kugel eine Spielfeldwand berührt hat. Wenn das der Fall ist wird die Geschwindigkeit vel_x und vel_y mit dem Faktor -1 multipliziert. Dadurch bewegt sich die Kugel dann in die entgegengesetzte Richtung.

  - **setPosition(self, x, y)**
    Setzt die Position der Kugel auf die entsprechende x- und y- Koordinaten.

  - **isOverlapping(self, other_ball)**
    Wenn die schwarze Kugel von einer anderen Kugel getroffen wird, wird hiermit ermittelt, ob sich die Kugeln „überlappen“. Wenn die Kugeln sich überlappen wird ein True zurückgegeben.

  - **resolveCollision(self, other_ball)**
    Die Kollisionsberechnung ist sehr kompliziert und wird hier nicht im Einzelnen erklärt. Siehe physikalische Impulsberechnung in Wikipedia oder anderswo. In dieser Methode wird die Kollision zweier Kugeln berechnet und entsprechend eine neue x- und y- Koordinate und Geschwindigkeit ermittelt

- **Klasse in Pocket.py**
  In dieser Klasse finden folgende Importe statt:
  - **import pygame**

  Folgend Methoden gibt es in dieser Klasse:

  - __init__(self, x, y, color, radius)

    Im Konstruktor werden die x- und y- Position gesetzt. Ebenso werden die Farbe und der Radius gesetzt.

  - **drawPocket(self, window)**
    Zeichnet mit dem importierten Packet pygame einen gefüllten Kreis.

- **Klasse in Queue.py**
  In dieser Klasse finden folgende Importe statt:
  - **import pygame**
    import math

  Folgend Methoden gibt es in dieser Klasse:

  - __init__(self, x, y, color, radius)

    Im Konstruktor werden die x- und y- Position gesetzt. Ebenso werden die Anfangslänge des Queues auf 100 und die Sichtbarkeit auf True gesetzt.

  - **createQueue(self, window)**
    Zeichnet mit dem importierten Packet pygame einen gefüllten Kreis.
  - **drawQue(self, window)**
    Wenn die Eigenschaft des Queues sichtbar == True ist wird mit Hilfe von dem Packet pygame eine Linie die 5 breit ist gezeichnet. Diese Linie ist dann das Queue.

Herausforderungen:

Die Herausforderungen waren das Umsetzen der physikalischen Impulsgesetze. Mir ist das größtenteils gelungen. Allerdings gibt es da noch einige Fehler, die zeitweilig auftreten und dies konnte ich leider in Anbetracht der knappen Zeit nicht beheben.

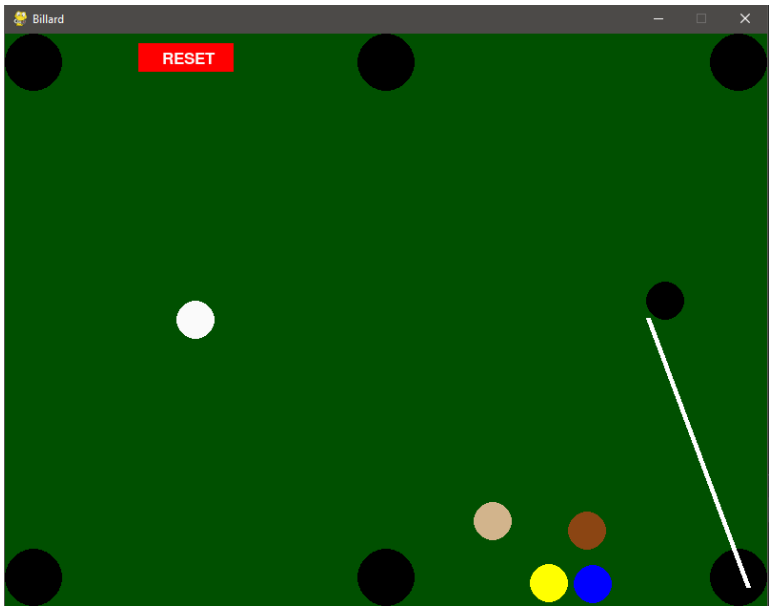
Eine weitere Herausforderung für mich waren die Klassen und Funktionen in eine extra Datei zu speichern. Das hatte nicht auf Anhieb funktioniert.

Das Billard Spiel unterliegt diversen physikalischen Gesetze. Es fiel mir nicht besonders leicht all diese Gesetze in Code umzusetzen – aber ich denke es ist mir größtenteils gelungen.

Probleme:

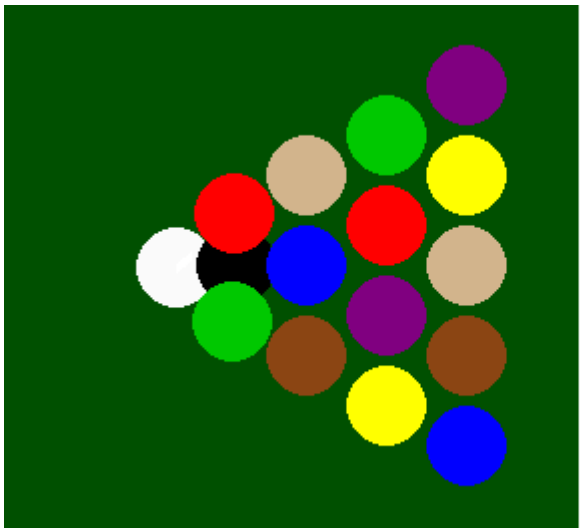
- Die Kollisionen funktionieren nicht perfekt. Manchmal kommt es vor, dass sich Kugeln überlappen. Wann das geschieht hatte ich nicht herausfinden können, dafür war die Zeit zu knapp. Dann sieht es in etwa so aus, der weiße Strich in dieser Abbildung ist das Queue:

- Es kann auch vorkommen, dass das Queue sich von der weißen Kugel löst und am Spielfeldrand erscheint. Das ist aber relativ selten der Fall. Warum das manchmal so auftritt hatte ich nicht herausgefunden.



- Auch kann es vorkommen, dass eine Kollision nicht vernünftig verarbeitet wird. Dann sieht es so aus, dass die Kugel in eine anderen Richtung rollt als sie angestoßen wurde. Auch das kommt selten vor.

- Manchmal funktioniert der erste Anstoß nicht richtig und die weiße Kugel bleibt in dem bunten Kugelhaufen stecken. Dabei funktioniert die Kollisionsberechnung nicht richtig. Ich habe nicht herausfinden können woran das lag.



- Es kann vorkommen, dass beim Einspielen der schwarzen Kugel, wenn diese die letzte Kugel ist, es nicht erkannt wird, dass man gewonnen hat. Dann muss man die schwarze Kugel eben noch einmal einspielen und erhält daraufhin die „Gewonnen“ Information.