Arcade-Spielautomat

Interdisziplinäre Projektarbeit

Marvin Büeler

Berufsbildungszentrum Goldau

Berufsmittelschule BM17/20a

24. Januar 2020

Experte: A. Bingisser

Vorwort

In der vorliegenden Arbeit geht es um die Entwicklung eines eigenen Videospiels. Ich habe mich für dieses Thema entschieden, weil ich mich seit ich klein bin für Computerspiele und ihre Entwicklung interessiere und schon seit einiger Zeit selbst ein Spiel programmieren wollte. Zudem möchte ich nach Abschluss meiner Lehre auf den Bereich der Informatik wechseln und auch Informatik studieren.

Die interdisziplinäre Projektarbeit in der Berufsmatura hat mir die Gelegenheit eröffnet, dieses Projekt in Angriff zu nehmen. Damit die Arbeit auch mit meiner Lehre in Verbindung gebracht werden kann, habe ich mich dazu entschieden einen Spielautomaten zu bauen. So benötige ich für das Projekt zusätzlich handwerkliches Geschick und muss den Automaten verkabeln.

Inhaltsverzeichnis

1 Einleitung

Für meine Arbeit setzte ich mir das Ziel, selbstständig ein Spiel im Retro-Stil zu entwickeln und aus Holz einen Spielautomaten zu konstruieren. Da ich in beiden dieser Bereiche nur wenig bis keine Erfahrung hatte, musste ich mir das notwendige Wissen zunächst aneignen.

Diese Dokumentation soll aufzeigen, wie ich vorgegangen bin, um diese Fähigkeiten zu erlernen und anzuwenden. Des Weiteren zeigt sie auf, welche verschiedenen Möglichkeiten es gibt, um diesen Lernprozess selbständig zu erzielen. Zudem erläutere ich die Überlegungen, die vor dem Beginn der Spieleentwicklung getätigt wurden und für die weitere Arbeit von Bedeutung waren. Dies beinhaltet sowohl die Auswahl der Programmiersprache, das Konzept des Spiels wie auch dessen Design.
In einem weiteren Schritt erkläre ich, wie ich bei der Konstruktion des Automaten vorgegangen bin. Dazu gehört die Planung des Automaten, die Auswahl der Materialien und ein Beschrieb, wie der Automat zusammengebaut und verkabelt wurde.

Ausgehend davon, dass ich noch nie eine Software entwickelt habe, waren meine Erwartungen zu Beginn des Projekt bescheiden. Mir war es wichtig, dass ich am Ende meines Projekts einen funktionierenden Spielautomaten präsentieren kann.

2 Arcade Spielautomat

Der Arcade Spielautomat wurde in den 80er und 90er Jahren populär und löste damals den Flipperkasten ab. Ursprünglich bezeichnete man ihn als Videospielautomaten und bekannt wurde er durch öffentliche Spielhallen in den USA. Die Spielhallen nannte man «Penny Arcades», weil für die damaligen Spielautomaten typisch war, dass man für das Spiel eine Münze einwerfen musste. «Computer Space» war der erste kommerzielle, münzbetriebene Arcade Spielautomat. Er wurde 1971 von Nolan Bushnell hergestellt, welcher später die Firma Atari gründete. Auch andere grosse Spiele Hersteller gewannen durch Videospielautomaten an Ansehen. Einer der bekanntesten ist Nintendo, welcher mit dem Arcade Spiel «Donkey Kong» seinen Durchbruch erlebte.

Es gibt viele verschiedene Varianten von Arcade Spielautomaten, aber die bekanntesten zwei Bauformen sind die aufrechten Automaten und der Cocktail- bzw. Tisch-Automat. Die Steuerung erfolgt bei beiden Varianten gleich. Meistens werden dafür Knöpfe eingesetzt und bei bestimmten Spielen zusätzlich ein Joystick. Der Unterschied der Automaten besteht darin, dass man bei dem aufrechten Automaten davorstehen muss, um auf den Monitor schauen zu können. Beim Cocktail-Automat hingegen setzt man sich und blickt auf einen Monitor, der in einer Tischplatte montiert ist. Der Cocktail-Automat wurde vor allem für Spiele verwendet, bei denen man gegeneinander spielt. Entsprechend ist das Spielfeld so aufgebaut, dass man sich gegenübersitzt. Eines der bekanntesten Spiele in dieser Art ist «Pong».

Mit dem Aufkommen der ersten Spielkonsolen gerieten die Arcade Spielautomaten und somit auch die zahlreichen Spielhallen in Vergessenheit. So ist das allerdings bei Videospielen. Alle Spiele sind nach einiger Zeit nicht mehr interessant, weil es immer wieder neuere gibt. Trotzdem haben die Spielautomaten noch nicht ganz ihren Wert verloren. Sie werden immer mehr als Sammlerstücke verkauft, sodass sich aus der Restauration von Spielautomaten eine kleine Nischen Industrie gebildet hat. Aus diesem erneuten Interesse an Arcade Spielautomaten sind auch vermehrt Heimwerk Projekte entstanden. Bastler bauen ihre eigenen Spielautomaten. Sie benutzen hierzu einen PC sowie einen Emulator, um die Spielhardware zu ersetzen. (The Verge, 2013)

Abbildung 2, Cocktail-Automat

Abbildung 1, Aufrechter-Automat

2.1 Eigener Spielautomat

Bei meinem eigenen Spielautomaten handelt es sich um keine der zwei üblichen Bauvarianten. Mir war es wichtig, dass mein Automat nicht viel Platz einnimmt, einfach transportierbar ist und dennoch mit einem aufrechten Automaten verglichen werden kann. Ich habe mich für einen Bartop-Arcade Automaten entschieden. Diese Bauform lässt sich am besten mit dem aufrechten Automaten vergleichen. Der einzige Unterschied in der Form ist die Höhe des Gestells. Man stellt ihn nicht auf den Boden, sondern er wird auf einem Tisch oder, wie es der Name sagt, auf einer Bar platziert.

Damit man einen so kompakten Spielautomaten selbst herstellen kann, ist ein Computer nötig, der ziemlich klein gehalten ist. Genau in diesem Fall eignet sich ein Raspberry Pi. Ein Raspberry Pi ist ein Einplatinencomputer und hat das Format einer Kreditkarte. Er ist nicht so leistungsstark wie ein Heimcomputer, aber eignet sich ausserordentlich gut für kleine Projekte wie meines. Gerade für den Einsatz in einem Arcade Spielautomaten wird er immer wieder von Personen empfohlen, die ihre Heimwerk Spielautomaten im Internet teilen.

Abbildung 3, Raspberry Pi

3 Spieleentwicklung

3.1 Quellcode

Jedes Programm, das von einem Rechner ausgeführt werden kann, besitzt einen Quellcode bzw. einen sogenannten Programmcode. In der Informatik versteht man darunter, einen in einer Programmiersprache verfassten, für den Menschen lesbaren Text. Der Quellcode beschreibt exakt, wie ein Computerprogramm funktioniert. Ausserdem ist ein Rechner in der Lage den Quellcode automatisch in eine ausführbare Maschinensprache zu übersetzen und das Programmierte auszuführen.

Quelltexte unterscheiden sich voneinander, aber bestimmte Elemente sind regelmässig anzutreffen. Dazu gehören **Befehle**. Ein Befehl ist eine Anweisung, die der Programmierer seinem Programm gibt, damit es weiss, was es zu tun hat. Weiter werden immer **Variablen** benutzt. Eine Variable ist nichts anderes, als eine Bezeichnung für etwas. **Vergleiche** werden ebenfalls ständig eingesetzt. Ein Vergleich ist eine logische Funktion. Mit ihr wird etwas verglichen und sofern dieser Vergleich zutrifft, wird ein Ereignis ausgelöst. Wichtig sind auch **Schleifen**. Eine Schleife ist ein Teil im Programm, der solange ausgeführt wird, bis dem Programm befohlen wird, es solle die Schleife verlassen. Zuletzt sind noch die **Kommentare**. Sie ermöglichen dem Verfasser des Quellcodes mithilfe von Text, den Quellcode zu beschreiben.

Für meine Arbeit habe ich einen eigenen Quellcode verfasst, der mein Spiel beschreibt. Wird dieser von einem Rechner ausgeführt, kann mein Arcade Spiel gespielt werden. (Ionos, 2018)

3.2 Programmiersprache

Um einen Quellcode zu erstellen, ist eine Programmiersprache nötig. Sie ermöglicht die Verständigung zwischen Mensch und Computer. Es gibt viele verschiedene Programmiersprachen, deshalb muss man sich für eine entscheiden. Bei der Auswahl muss darauf geachtet werden, dass der verwendete Rechner auch über ein Programm verfügt, dass die verwendete Programmiersprache übersetzen kann. Findet man auf der Festplatte z.B. eine Datei mit der Endung «.cpp», weist dies darauf hin, mit welchem Programm die Datei geöffnet wird und in welcher Programmiersprache der Quellcode verfasst wurde. In diesem Fall wäre das die Programmiersprache C++.

Da ich mich bereits dafür entschieden hatte, einen Raspberry Pi als Computer für meinen Spielautomaten zu verwenden, achtete ich bei der Auswahl der Programmiersprache, dass sie vom Raspberry Pi gelesen werden kann. Python ist eine Programmiersprache, die vom Standard Raspberry Pi OS ohne weitere Programme gelesen werden kann. Zudem wird Python für Lernende empfohlen, weil sie gut lesbar ist. Für Python wurde sogar eine Programmbibliothek erstellt, die das Programmieren von Videospielen vereinfacht. Die Programmbibliothek Pygame enthält Module zum Abspielen und Steuern von Grafiken und Sounds sowie zum Abfragen von Eingabegeräten. (Ionos, 2018)

3.3 Programmierumgebung

Um einen Quellcode zu schreiben, reicht ein einfacher Texteditor schon aus. Es gibt aber Entwicklungsumgebungen, kurz IDE (Integrated Development Environment), die die Arbeit für die Erstellung von Programmen vereinfachen. Sie enthalten wichtige Tools für das Verfassen von Software. Zu diesen Tools zählt beispielsweise Syntaxhervorhebung. Das heisst, dass bestimmte Befehle verschieden farbig gekennzeichnet

werden. Weiter besitzen IDEs meistens einen Compiler, der es ermöglicht den Programmcode, ohne abzuspeichern auszuführen.

In meinem Fall habe ich mich für die IDE PyCharm entschieden. Sie ist kostenlos und ist visuell einfach gestaltet. Zudem verfügt sie bereits über einen Compiler, der Python Quellcodes ausführen kann.

3.4 Aneignung von Grundwissen

Bisher hatte ich nur wenig Erfahrung mit dem Erstellen von Programmcodes. Das einzige Wissen, dass ich zu Beginn meiner Projektarbeit nachweisen konnte, lernte ich in der Berufsschule. Dazu gehört ein bereichsübergreifendes Projekt, bei dem mit Hilfe eines Arduino Mikrocontrollers ein selbstfahrendes Auto realisiert wurde. Jedoch besteht ein grosser Unterschied zum Programmieren eines Computerspiels. Trotzdem war dieses Vorwissen nützlich, da ich bereits zum Teil das Verständnis hatte, wie ein Computer einen Programmcode abarbeitet.

Damit ich aber mit der Spieleentwicklung beginnen konnte, gehörte zunächst eine Lernphase dazu, bei der ich die Grundbefehle der Spieleentwicklung mit Python und Pygame kennenlernte. Zu meinem Glück bin ich auf eine hilfreiche Video Tutorial-Reihe[1] gestossen, welche ich während zwei Wochen erarbeitet habe. Ich wiederholte fortlaufend alle Schritte, die in den Videos erklärt wurden. Nach dem letzten Tutorial hatte ich ein kleines Prototyp-Spiel, welches mehrere Grundfunktionen für ein Computerspiel beinhaltete.

Bevor ich aber mit meinem eigenen Spiel begann, ging ich die Befehle dieses Beispiels nochmals durch und suchte auf verschiedensten Websites nach möglichen Lösungen, wie ich den Tutorial Quellcode verbessern konnte. Besonders hilfreich waren hierbei Internet Foren. Ziel dieser zusätzlichen Recherche war die Aneignung von weiterem Wissen, damit ich dieses bei meinem eigenen Spiel umsetzen konnte.

3.5 Spiel Konzept

Nachdem ich mit dem Prototyp-Spiel abgeschlossen hatte, ging es weiter mit dem nächsten Schritt. Es war an der Zeit mir Gedanken zu meinem eigenen Spiel zu machen. Ich habe bewusst damit gewartet, bis ich über genug Wissen verfügte, um mir vorstellen zu können, was im Bereich des Möglichen liegt. Anhand des Gelernten konnte ich nun ein Spiel Konzept ausarbeiten.

Bei meinem Spiel liess ich mich von früheren Arcade-Spielen inspirieren. Typische Beispiele von Arcade-Spielen, die auch auf Automaten ausgeführt wurden, sind der Jump'n'Run «Donkey Kong», das Maze-Spiel «Pac-Man» oder der Shmup «Space Invaders». Das Prinzip von «Space Invaders» hat mich auf Anhieb angesprochen und es hat den Eindruck erweckt, dass ich es selbst zu entwickeln vermochte.

Ich habe mich folglich stärker über das Spiel-Genre Shmup informiert. Shmup ist eine Abkürzung und steht für Shoot'em'up. Bei dieser Art von Spiel geht es darum, dass fortlaufend Gegner auftauchen, die eliminiert werden müssen. Es gibt sehr viele verschiedene Spiele, die auf diesem Konzept aufgebaut sind, aber trotzdem sind alle auf ihre Art einzigartig. Infolgedessen habe ich versucht, ein Spiel auf dieser Basis zu entwickeln.

[1] (YouTube, 2017), Tech With Tim

Um eine erste Idee zu erlangen, skizzierte ich, wie mein Spiel in etwa aussehen sollte und was darin alles enthalten sein sollte. Ich habe mich dafür entschieden, dass mein Spiel von unten nach oben verläuft und dass es zwei verschiedene Arten von Gegner gibt. Mithilfe der Skizze habe ich mir mein erstes Ziel für die Entwicklung des Spiels gesetzt. Jedoch war mir bewusst, dass ich, wenn nötig im Verlauf der Arbeit noch Änderungen vornehmen muss.

3.6 Programmierung des Spiels

Für die Programmierung erstellte ich eine Liste mit den Elementen, die ich in mein Spiel einbeziehen wollte. Eine entsprechende To-Do Liste ist eine gute Möglichkeit, um ständig den Überblick des eigenen Fortschritts zu haben. Die Entwicklung habe ich in drei Etappen aufgeteilt. Hierzu habe ich mich entschieden, damit sichergestellt werden konnte, dass sich meine Ideen umsetzen lassen. Beim Programmieren ist es wichtig, dass man in kleinen Schritten vorgeht. Man fügt Schritt für Schritt Funktionen hinzu und kontrolliert laufend, ob alles (noch) funktioniert. Das habe ich bereits bei meinem Schulprojekt und erneut in der Tutorial Reihe erfahren.

Um noch klar zu stellen, ich werden im folgenden Abschnitt vor allem auf die Aspekte meines Quellcodes eingehen, die erklären, wie mein Spiel aufgebaut ist. Natürlich braucht es noch wesentlich mehr, damit das Programm von einem Rechner ausgeführt werden kann. Wer sich für den kompletten Programmcode interessiert, findet diesen im Anhang dieses Dokumentes.

3.6.1 Grundlage

Die Grundlage meines Spiels bilden das Spielfeld, auf dem man sich bewegen kann, die Spielfigur und eine erste Form von Gegnern.

3.6.1.1 Spieler-Sprite

Begonnen habe ich mit der Spielfigur. Für den Anfang war es mein Ziel, mit Hilfe der Pfeiltasten auf der Tastatur ein Rechteck in alle vier Richtungen bewegen zu können. Dazu habe ich das Sprite-Klassen Modul von Pygame benutzt. Bevor ich aber weiter darauf eingehe, erläutere ich, was Sprites sind. In einem Videospiel werden alle Objekte, die sich bewegen oder mit denen man interagieren kann, als Sprites bezeichnet. Es handelt sich dabei Grafikobjekte, die über das Hintergrundbild eingeblendet werden.

Das Sprite-Klassen Modul von Pygame, vereinfacht das Erstellen von Sprites. Es stellt eine Basis zur Verfügung, die für die meisten Sprites bereits ausreicht. Es beinhaltet Funktionen, die die Grösse, Farbe, Form usw. eines Sprites bestimmen.

Der Folgende Quellcodeabschnitt zeigt das Beispiel meiner Spielfigur und wie die Visualisierung davon aussehen würde. In diesem Abschnitt gebe ich an, wie gross meine Spielfigur sein und wie sie eingefärbt werden soll. Die beiden Variablen **speedx** und **speedy** werden im nächsten Schritt gebraucht.

```
class Player(pygame.sprite.Sprite): #Erstellt den Sprite für den Spieler
    def __init__(self): #Repräsentiert die Instanz
        pygame.sprite.Sprite.__init__(self) #Pygame Modul
        self.image = pygame.Surface((70,50)) #Grösse des Objekt in Pixel
        self.image.fill(GREEN) #Farbe des Objekts
        self.rect = self.image.get_rect() #Fragt die Koordinaten des Objekts ab
        self.speedx = 0 #Variable Geschwindigkeit x-Achse
        self.speedy = 0 #Variable Geschwindigkeit y-Achse
```

Abbildung 4, Programmcode Spieler Sprite

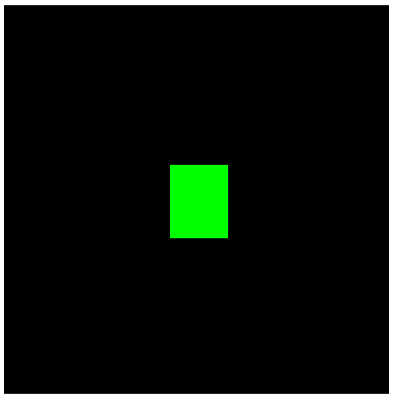


Abbildung 5, Visualisierung Spieler Sprite

Um dieses Rechteck nun auch mit den Pfeiltasten bewegen zu können, muss eine Update-Definition der Spieler-Sprite Klasse hinzugefügt werden. Diese Update-Definition wird immer wieder vom Programm abgefragt. In ihr wird kontrolliert, ob eine Pfeiltaste gedrückt wird. Trifft das zu, dann werden die Koordinaten des Objekts geändert. Im folgenden Beispiel zeige ich auf, wie das für die Pfeiltaste nach oben aussieht.

```
class Player(pygame.sprite.Sprite): #Erstellt den Sprite für den Spieler
    def __init__(self):...

    def update(self): #Methode zur Steuerung des Sprite-Verhaltens
        self.speedx = 0 #Zurücksetzen der Geschwindigkeit
        self.speedy = 0 #Zurücksetzen der Geschwindigkeit
        keystate = pygame.key.get_pressed() #Abfrage der Tasteneingabe
        if keystate[pygame.K_UP]: #Kontrolle, ob Pfeiltaste "Hoch" gedrückt wird (Wenn)
            self.speedy = -5 #Ändern der Geschwindigkeits Variable (Dann)
        self.rect.y += self.speedy #Position des Rechtecks verändern
```

Abbildung 6, Programmcode Spieler Update-Definition

3.6.1.2 Grösse des Spielfeldes

Beim Skizzieren meines Spiels habe ich das Spielfeld auf der linken und rechten Seite begrenzt. Die Spielfigur hat folglich nicht die Möglichkeit, sich bis an den Rand zu bewegen. Um das zu erreichen, habe ich der Spielfigur nicht eine Wand als Grenze gesetzt, sondern ich habe in der Update-Definition bestimmt, wie weit die Spielfigur sich an den Rand bewegen kann. Auch wenn die Pfeiltaste weiterhin gedrückt wird, die Koordinaten der Spielfigur werden immer wieder auf einen bestimmten Wert zurückgesetzt, falls die Grenze überschritten wird. In meinem Fall liegt die Grenze bei 200

Pixel von der linken Seite und bei 200 Pixel von der rechten Seite. Ich habe den transparenten Rand rot markiert, um das Beispiel besser darzustellen.

```
def update(self): #Methode zur Steuerung des Sprite-Verhaltens
    ...
    if self.rect.right > width -200: #Vergleicht Rechte Spielerkoordinate mit Grenzwert
        self.rect.right = width -200 #Setzt Koordinate auf den Grenzwert
    if self.rect.left < 200: #Vergleicht linke Spielerkoordinate mit Grenzwert
        self.rect.left = 200 #Setzt Koordinate auf den Grenzwert
```

Abbildung 7, Programmcode Spielergrenze

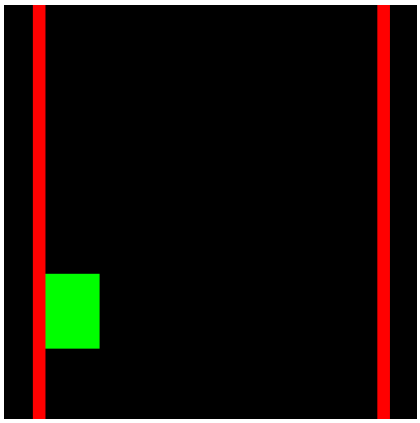


Abbildung 8, Visualisierung Spielergrenze

3.6.1.3 Gegner Sprite

Da mein Spiel später von unten nach oben Verlaufen sollte, müssen sich die gegnerischen Sprites von oben nach unten bewegen. Der Erstellung des Gegner-Sprites erfolgt wie die Spielerfigur. Der Unterschied der beiden liegt in der Update-Definition. Der Gegner wird nicht über Tasten bewegt, sondern hat eine vorgeschriebene Geschwindigkeit, die nicht verändert wird. Ich setzte also die **speedy** Variable des Gegners auf einen bestimmten Wert, mit dem er fortlaufend immer von oben nach unten fährt. Da sich der Gegner nicht auf der x-Achse bewegt, muss keine Grenze bestimmt werden. Ich sage dem Programm lediglich, in welchem Bereich die Gegner erzeugt werden können. Dieser Bereich ist später aber nicht auf dem sichtbaren Spielfeld. Zum besseren Verständnis habe ich die Bewegungen der Sprites mit Pfeilen hervorgehoben und den Bereich markiert, in dem die Gegner auftauchen.

```
class Enemy(pygame.sprite.Sprite): #Erstellt den Sprite für den Spieler
    def __init__(self): #Repräsentiert die Instanz
        pygame.sprite.Sprite.__init__(self) #Pygame Modul
        self.image = pygame.Surface((50,30)) #Grösse des Objekt in Pixel
        self.image.fill(BLUE) #Farbe des Objekts
        self.rect = self.image.get_rect() #Fragt die Koordinaten des Objekts ab
        self.speedy = 2 #Variable Geschwindigkeit y-Achse
        self.rect.x = rn.randrange(200, width - 200 - self.rect.width) #x-Koordinaten des Spawnbereich
        self.rect.y = rn.randrange(-500, -300) #y-Koordinaten des Spawnbereich

    def update(self): #Methode zur Steuerung des Sprite-Verhaltens
        self.rect.y += self.speedy #Lässt Objekt nach unten Fahren
```

Abbildung 9, Programmcode Gegner-Sprite

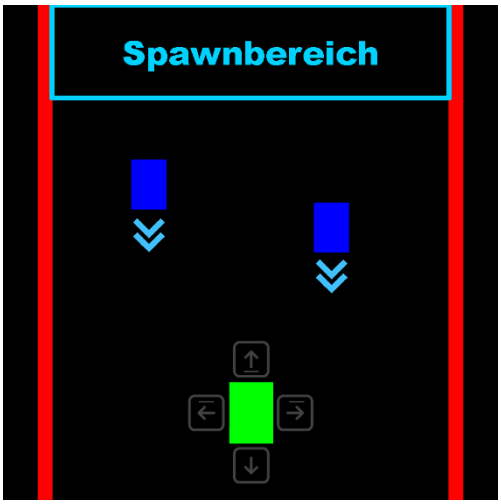


Abbildung 10, Visualisierung Spielgrundlage

3.6.2 Funktionen

Durch das Hinzufügen von Funktionen, entsteht aus der Grundlage ein Spiel. Für mein Spiel plante ich mehrere Funktionen. Der Spieler darf nicht den Gegner berühren, ansonsten ist das Spiel beendet. Weiter hat der Spieler die Möglichkeit, mit einem Taster einen Schuss abzufeuern, der eine zweite Form von Gegner zerstören kann. Durch die Zerstörung dieser Gegner bekommt der Spieler Punkte. Ziel ist es, möglichst viele Punkte zu erzielen, ohne dabei mit einem gegnerischen Sprite in Berührung zu kommen. Damit das Ganze aber noch mehr Spielspass bereitet, nimmt die Schwierigkeit des Spiels mit der Punktzahl zu. Dies geschieht, indem der Spielverlauf immerzu beschleunigt.

3.6.2.1 Kollision von Spieler mit Gegner

Begonnen habe ich mit der Kollision der verschiedenen Sprites. Da diese Funktion ständig überwacht wird, muss diese Überwachung im sogenannten **Loop** meines Quellcodes erfolgen. Als Loop wird eine Schleife im Programmcode bezeichnet, die ständig ausgeführt wird. Ausser man gibt im Loop die Anweisung, dass ein andere Teil ausgeführt werden soll.

Für die Detektion zweier Sprites stellt die Pygame-Bibliothek ein Modul zur Verfügung. Mit dem **Spritecollide** Befehl wird kontrolliert, ob sich zwei ausgewählte Sprites berühren. Da meine Sprites später keine Rechtecke sein sollen, muss ich noch eine **Hitbox** definieren. Eine Hitbox ist der physische Bereich von einem Sprite. Ich benutze eine kreisförmige Hitbox. Bei den meistens Arcade-Spielen werden kreisförmige benutzt, weil mit dieser Form die meisten Flächen einfach abgedeckt werden können. Ich habe sie für die Visualisierung rot eingefärbt und über die Sprites gelegt.

```
class Player(pygame.sprite.Sprite): #Erstellt den Sprite für den Spieler
    def __init__(self): #Repräsentiert die Instanz
        pygame.sprite.Sprite.__init__(self) #Pygame Modul
        ...
        self.radius = 28 #Grösse der Hitbox, Radius des Kreises

    def update(self): #Methode zur Steuerung des Sprite-Verhaltens
        ...

class Enemy(pygame.sprite.Sprite):  # Erstellt den Sprite für den Spieler
    def __init__(self):  # Repräsentiert die Instanz
        pygame.sprite.Sprite.__init__(self)  # Pygame Modul
        ...
        self.radius = 25  # Grösse der Hitbox, Radius des Kreises

    def update(self):  # Methode zur Steuerung des Sprite-Verhaltens
        ...
```

Abbildung 11, Hitbox definieren

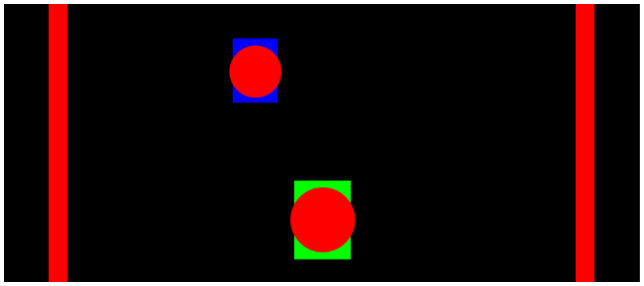


Abbildung 12, Hitbox der Sprites

Als Zusatz habe ich definiert, dass die Spielfigur eliminiert werden soll, sofern eine Kollision detektiert wird.

```
#----------Game Loop----------#
run = True #Loop Variable
while run: #Schleifen Bedingung
    ...
    hits = py.sprite.spritecollide(Player, Enemy, True, py.sprite.collide_circle) #**
    #Kollision detektieren**#
    if hits: #Wenn Kollision zutrifft dann:
        Player.kill() #Spielfigur eliminieren
    ...
```

Abbildung 13, Programmcode Spritecollide

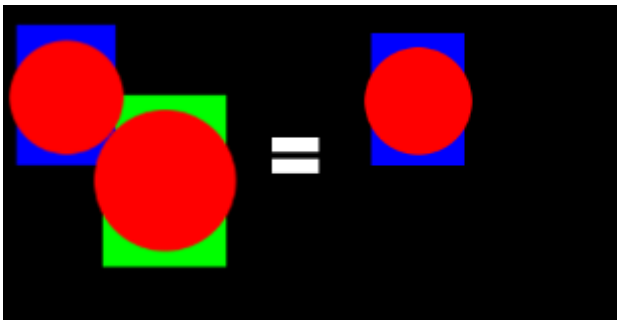


Abbildung 14, Visualisierung der Kollision

3.6.2.2 Schussfunktion

Damit die Spielfigur nicht nur hin und her fahren kann, habe ich eine Schussfunktion hinzugefügt. Sie macht das Spiel noch einmal spannender, weil das Hauptziel ist, möglichst viele Gegner mit den Schüssen zu treffen. Die Schussfunktion habe ich auch mit Hilfe des Sprite-Klassen Modul gelöst. Die Schüsse, die die Spielfigur abfeuert, sind nichts anderes, als weitere Sprites. Sie treten allerdings nicht im Verlauf des Spiels einfach auf, sondern werden per Knopfdruck erstellt. Das Verhalten der Schüsse gleicht demjenigen der gegnerischen Sprites. Der einzige Unterschied liegt darin, dass sie sich von unten nach oben bewegen. Die Schuss-Sprite Klasse sieht wie folgt aus.

```
class Bullet(pygame.sprite.Sprite): #Erstellt den Sprite für den Spieler
    def __init__(self,x,y): #Repräsentiert die Instanz
        py.sprite.Sprite.__init__(self) #Pygame Sprite Modul
        self.image = pygame.Surface((20,5)) #Definiert Grösse
        self.image.fill(WHITE) #Einfärben des Sprites
        self.rect = self.image.get_rect() #Abfragen der Koordinaten
        self.radius = 8 #Hitbox Grösse definieren
        self.rect.bottom = y #y-Koordinate von Start bestimmen
        self.rect.centerx = x #x-Koordinate von Start bestimmen
        self.speedy = -3 #Geschwindigkeit des Objekts

    def update(self): #Methode zur Steuerung des Sprite-Verhaltens
        self.rect.y += self.speedy #Bewegung des Schusses
        if self.rect.bottom < 0: #Wenn nichts getroffen wird,
            self.kill()          #wird der Schuss gelöscht
```

Abbildung 15, Programmcode Schuss-Sprite

Damit man nun den Schuss mit einer Taste abfeuern kann, muss man die Update-Definition von der Spieler-Sprite erweitern. Genau wie bei der Bewegung bestimmt man eine Taste und wenn diese gedrückt wird, soll ein Schuss-Sprite erstellt werden. Damit dieser nicht einfach irgendwo auf dem Bild auftaucht, gibt man noch die y- und x-Koordinate an. Ich habe für diese die Koordinaten genommen, die die Spielfigur hat und das Programm selbständig abfragen kann. Für die Taste der Schiessfunktion habe ich vorläufig die Leertaste gewählt.

```
def update(self): #Methode zur Steuerung des Sprite-Verhaltens
    keystate = pygame.key.get_pressed() #Abfrage der Tasteneingabe
    ...
    if keystate[pygame.K_SPACE]: #Kontrolle ob Leertaste gedrückt wird
        Bullet(self.rect.centerx, self.rect.top) #Erstellen des Schusses mit
                                                 #x-,y-Koordinate in der Klammer
```

Abbildung 16, Programmcode erweiterte Update-Definition von Spielfigur

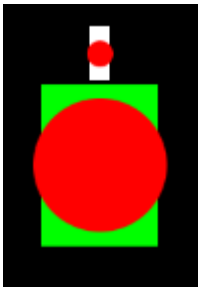


Abbildung 17, Visualisierung Schuss

Damit die Gegner nun auch getroffen werden können, kann man wiederum mit Hilfe des **Spritecollide-**Befehls die Kollision zwischen Schuss und Gegner detektieren. Zusätzlich wird bei festgestellter Kollision die Punktzahl um einen Punkt erhöht.

```
bullethits = py.sprite.spritecollide(Enemy, Bullet, True, True, py.sprite.collide_circle) #**
#Kollision detektieren**#
    for hit in bullethits: #Für jede Kollision erhöht
        score += 1         #sich die Punktzahl um 1
```

Abbildung 18, Programmcode Kollision zwischen Schuss und Gegner

Nachdem die Schussfunktion funktionierte, fügte ich sie auch dem gegnerischen Sprite hinzu. Somit konnte die Schwierigkeit des Spieles nochmals erhöht werden. Auslöser der Schussfunktion beim Gegner ist das Erreichen eines bestimmten Wertes der y-Koordinate. Erreicht wird das auch durch einen Vergleich, wie bei der Spielfigur. Nur vergleicht man nicht die Tasteneingabe, sondern die Position von dem Sprite mit dem vorgegebenen Wert. Als zweiten Gegner benutze ich ein Duplikat der Grundlage des gegnerischen Sprites. Er kann keine Schüsse abfeuern, dafür aber durch einen Schuss nicht zerstört werden. Ihm muss man ausweichen, um im Spiel weiter zu kommen.

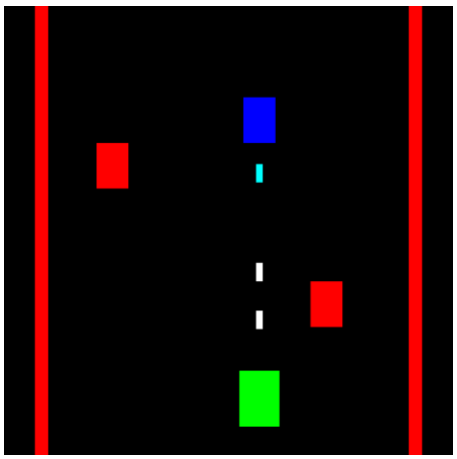


Abbildung 19, Visualisierung fertiges Spielkonzept

3.6.3 Steuerung

Da das ganze Spiel später nicht am Computer, sondern auf dem Spielautomaten gespielt werden soll, muss die Steuerung der Spielfigur über einen Joystick und Taster erfolgen. Zuvor habe ich alles so geregelt, dass das Spiel mit den Pfeiltasten und der Leertaste gespielt wird. Um das zu ändern, müssen nur die Bedingungen für die Bewegung der Spielfigur in der Update-Definition ausgetauscht werden.

Um heraus zu finden, wie die Bedingungen bei meinem Controller heissen, habe ich ein Beispielprogramm[2] von der offiziellen Pygame Webseite heruntergeladen. Dieses Beispielprogramm gibt über eine Anzeige aus, welche Taster von einem per USB angeschlossenen Controller gedrückt werden. Mein Joystick und Taster werden auch über ein Mittelstück mit einer USB Verbindung an den Computer angeschlossen.

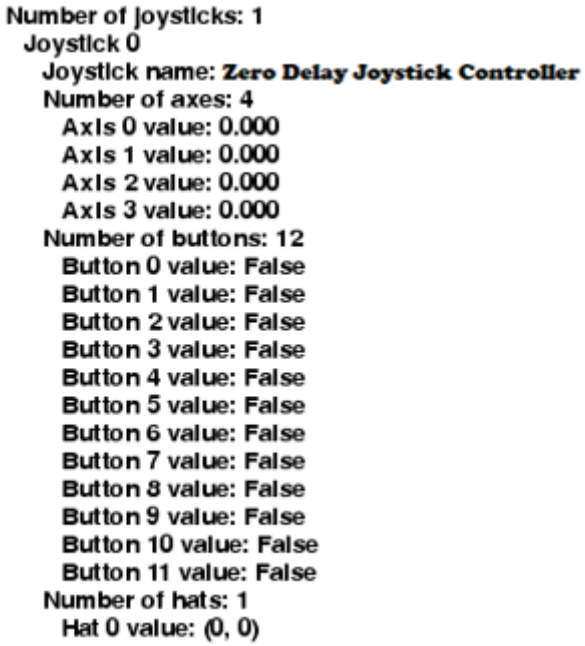


Abbildung 20, Anzeige Controller Programm

Ich habe folglich meinen Controller an den Computer angeschlossen und das Pygame Programm für die Auswertung gestartet. Anhand der Anzeige habe ich mir notiert, welche Werte sich verändern, wenn ich den Joystick und den Taster bediene. Mit den daraus gewonnenen Angaben konnte ich im Quellcode des Beispielprogrammes nachschauen gehen, wie die Werte angesteuert werden.

Als Beispiel verwende ich den Joystick, wenn er nach vorne gedrückt wird. Dies sollte in meinem Spiel bewirken, dass die Spielfigur nach oben fährt. Wenn ich den Joystick nach vorne drücke, verändert sich im Interface den Wert bei dem Titel **«Axis 1 value»**. Der Wert ändert von der Zahl 0 auf -1. Im Quellcode hielt ich also Ausschau, mit welcher Bedingung dieser Wert geändert wird. In diesem Fall ist dieser Teil des Programmcodes interessant:

[2] https://www.pygame.org/docs/ref/joystick.html

```
for i in range(axes):
            axis = joystick.get_axis(i)
            textPrint.tprint(screen, "Axis {} value: {:>6.3f}".format(i, axis))
        textPrint.unindent()
```

Abbildung 21, Ausschnitt Programmcode Controller Pygame

In diesem Abschnitt ist das grün geschriebene der Titel, bei dem sich der Wert verändert hat. Interessant für mein Problem ist aber der Teil mit der Variable **axis**. Nach dem Gleichheitszeichen steht nämlich der Befehl, den die Eingabe des Joysticks auswertet. Diesen habe ich in den Quellcode meines Spieles eingefügt, damit die Spielfigur mit dem Joystick bewegt werden kann. Ich konnte nachvollziehen, dass es die Achse Nr. 1 ist, weil sich dort der Wert verändert hat. Deshalb habe ich in meiner Update-Definition von dem Spieler Sprite eine Variable erstellt, die den Wert der Joystick Achse Nr. 1 vertritt. Sobald diese Variable einen Wert von -1 erreicht, soll sich die Spielfigur nach oben bewegen. Diese Bedingung habe ich mit der Pfeiltaste der Tastatur ausgetauscht und damit erreicht, dass der Joystick zur Steuerung verwendet werden kann. Rot umrahmt ist nachfolgend die abgeänderte Bedingung ersichtlich.

```
def update(self): #Methode zur Steuerung des Sprite-Verhaltens
    self.speedx = 0 #Geschwindigkeit zurücksetzen
    self.speedy = 0 #Geschwindigkeit zurücksetzen
    joystick = py.joystick.Joystick(0) #Bestimmen welcher Joystick verwendet wird
    joystick.init() #Joystick initialisieren, damit er verwendet werden kann
    achse_1 = joystick.get_axis(1) #Variable für die Joystickachse definieren
    if achse_1 == -1: #Kontrolle ob Variable den Wert -1 hat
        self.speedy = -5 #Geschwindigkeits Variable verändern
    self.rect.y += self.speedy #Position des Rechtecks verändern
```

Abbildung 22, Programmcode Ansteuerung-Joystick

3.6.4 Grafik und Audio

Damit ein Videospiel auch wirklich als Videospiel bezeichnet werden kann, muss es über ein eigenes Grafik und Audio Design verfügen. Bei den Grafiken habe ich mit Photoshop im Pixel Art Stil die einzelnen Sprites und den Hintergrund entworfen. Ich habe mit Absicht diesen verpixelten Stil gewählt, weil die typischen Arcade-Spiele von früher alle so dargestellt wurden. Das Einfügen der Grafiken erfolgte grundsätzlich einfach. Mit dem Sprite-Klassen Modul von Pygame ist es möglich, jedem einzelnen Sprite ein Bild anzufügen. Dazu muss nur der Pfad angegeben werden, in welchem das Bild auf dem Computer abgespeichert ist, um anschliessend bei der Erstellung des Sprites das entsprechende Bild auszuwählen.

```
img_dir = path.join(path.dirname(__file__), 'img')
#Pfad des Ordners mit den Grafiken


class player(py.sprite.Sprite): #Erstellt den Sprite für den Spieler
    def __init__(self): #Repräsentiert die Instanz
        py.sprite.Sprite.__init__(self) #Pygame Sprite Modul
        self.image = player_img[0] #Bezeichnung der Grafik
        ...
```

Abbildung 23, Programmcode Grafik anfügen

Abbildung 24, Spieldesign

Die Sounds, die im Spiel vorkommen, habe ich mit Hilfe eines Programms erzeugt, das speziell darauf ausgelegt ist 8-Bit Geräusche zu generieren. Das Programm heisst BFXR und ist kostenlos. 8-Bit Geräusche sind auch typisch für Arcade-Spiele, folglich habe ich mich erneut von früheren Spielen inspirieren lassen. Die Spieltöne werden gleich wie die Grafiken in das Programm eingefügt. Nur habe ich die Töne nicht einem Sprite zugeordnet, sondern lasse sie immer zu bestimmten Zeitpunkten, wie z.B. beim der Kollision von zwei Sprites, abspielen.

```
snd_dir = path.join(path.dirname(__file__), 'snd')
#Pfad des Ordners mit den Tönen

bullethits = py.sprite.spritecollide(Enemy, Bullet, True, True, py.sprite.collide_circle) #**
    #Kollision detektieren**#
        for hit in bullethits: #Für jede Kollision erhöht
            score += 1         #sich die Punktzahl um 1
            explosion_enemy_sound.play() #Abspielen des Soundeffektes
```

Abbildung 25, Programmcode Soundeffekt

3.6.5 Vorgehen bei Problemen

Bei der Programmierung bin ich ständig auf Probleme gestossen. Oftmals wenn etwas Neues hinzugefügt wurde, hat diese neue Funktion nicht funktioniert. In diesem Fall ist die beste Lösung im Internet nach dem Problem zu suchen, in der Hoffnung, dass jemand anderes schon einmal auf das gleiche Problem gestossen ist. So war es zumindest bei mir meistens der schnellste Weg ein Problem zu lösen. Viele Male analysierte ich auch Quellcodes von anderen Programmen, wie ich es z.B. bei dem Joystick angegangen bin. Sofern man weiss, nach was man sucht, ist es hilfreich Programmcodes von anderen Programmen lesen und verstehen zu können. So hat man die Möglichkeit Teile von anderen Programmen zu adaptieren und in das eigene einzubauen.

Häufig kam es auch vor, dass etwas bereits Vorhandene durch das Hinzufügen einer neuen Funktion nicht mehr funktioniert hat. Bei diesem Problem habe ich versucht, den Teil der neu dazu gekommen ist, auf eine andere Weise zu realisieren. Mehrfach musste ich aber auch in diesen Fällen nach Hilfe auf verschiedenen Websites suchen. Besonders hilfreich war die Open Source Website «GitHub» und das Internet Forum «Stack Overflow». Auf beiden Websites findet man Beiträge, welche diverse Probleme in der Programmierung adressieren.

4 Automat Gestell

4.1 Konstruktion

Das Gestell meines Spielautomaten habe ich mit der CAD Software Fusion 360 von Autodesk konstruiert. Fusion 360 ermöglicht es Bauteile und entsprechend die Konstruktion als 3D-Design zu entwerfen. Es hat eine einfache Bedienung und ist für Studenten kostenlos nutzbar. Die Konstruktion erfolgte innerhalb kürzester Zeit, da ich bei dem Design meines Spielautomaten möglichst schlicht bleiben wollte. Diese Entscheidung habe ich aus dem Grund getroffen, weil ich nicht über das nötige Werkzeug verfüge, um technisch komplexe Konstruktionen zu bauen. Zusätzlich war ein Punkt bei meiner Konstruktion, dass das Material möglichst kosten günstig ist. Das Material habe ich bereits vor dem Beginn der Konstruktion ausgewählt, weil es eine wichtige Rolle spielt, welche Rohmasse das Material hat.

Das grösste Bauteil, welches in meinen Automaten eingebaut ist, ist der Monitor. Deshalb hatte die Grösse des Monitors einen Einfluss auf meine Konstruktion. Mein Spielautomat ist klein gehalten, aber gerade genug gross, damit der Monitor gut darin Platz findet. Damit ich mir ein Bild meiner Vermassung machen konnte, habe ich nach meinem ersten Entwurf einen Prototypen aus Karton gebastelt. Mit diesem Vorgehen konnte ich mir gut vorstellen, wie der Automat später aussehen wird. Der Prototyp hat mich sogleich überzeugt, weshalb ich mit meinem ersten Entwurf auch gleich weiterfuhr.

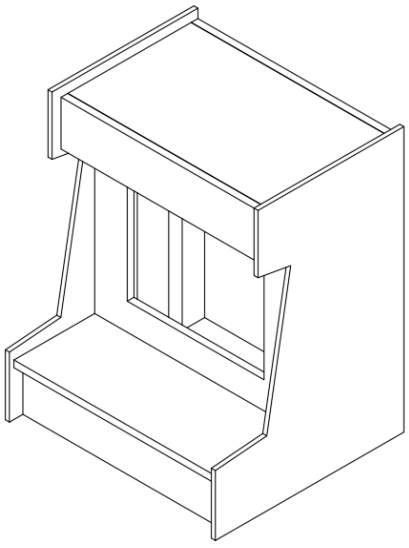


Abbildung 26, 3D-Modell Automat

4.2 Aufbau

Das Gestell ist aus Holz, welches ich im Baumarkt eingekauft habe. Die grossen Platten habe ich auch gleich dort zuschneiden lassen, da es mir wichtig war, dass die gerade zugeschnitten sind. Alles andere, was zum Aufbau gehört habe ich selbst gemacht. Die Klappe und das Aluminiumgitter auf der Rückseite sind nicht auf meiner Zeichnung ersichtlich. Diese zwei Elemente habe ich ohne Planung hinzugefügt.

Begonnen habe damit, alle Ausschnitte und Bohrungen zu machen. Die Ausschnitte habe ich mit der Stichsäge ausgesägt. Mit einer Tischsäge wären die Kanten bestimmt gerader worden, aber ich besitze leider keine und mit dem Ergebnis bin ich auch so zufrieden. Leider bemerkte ich, dass das ausgewählte Holz für spanende Arbeiten mit

der Stichsäge nicht optimal ist. Auf der Rückseite der Schnittkante fing das Holz an auszubrechen. Ich habe versucht mit Holzpaste diese Fehler auszubessern.

Abbildung 27, Kantenreparatur mit Holzpaste

Auch bei den Bohrungen ist das gleiche passiert. Bei der ersten Bohrung habe ich dann ein zweites Holzstück dahintergeklemmt. Das hat aber das Ausbrechen nicht komplett vermieden. Weil die zweite Bohrung besser sichtbar ist als die erste, habe ich diese in meinem Lehrbetrieb an der Standbohrmaschine, anstatt mit dem Akkubohrer gebohrt. Ich habe auch einen Stufenbohrer anstatt eines Spiralbohrers verwendet. Dieses Vorgehen hat sich bewährt. Bei der zweiten Bohrung ist das Holz unbeschädigt geblieben.

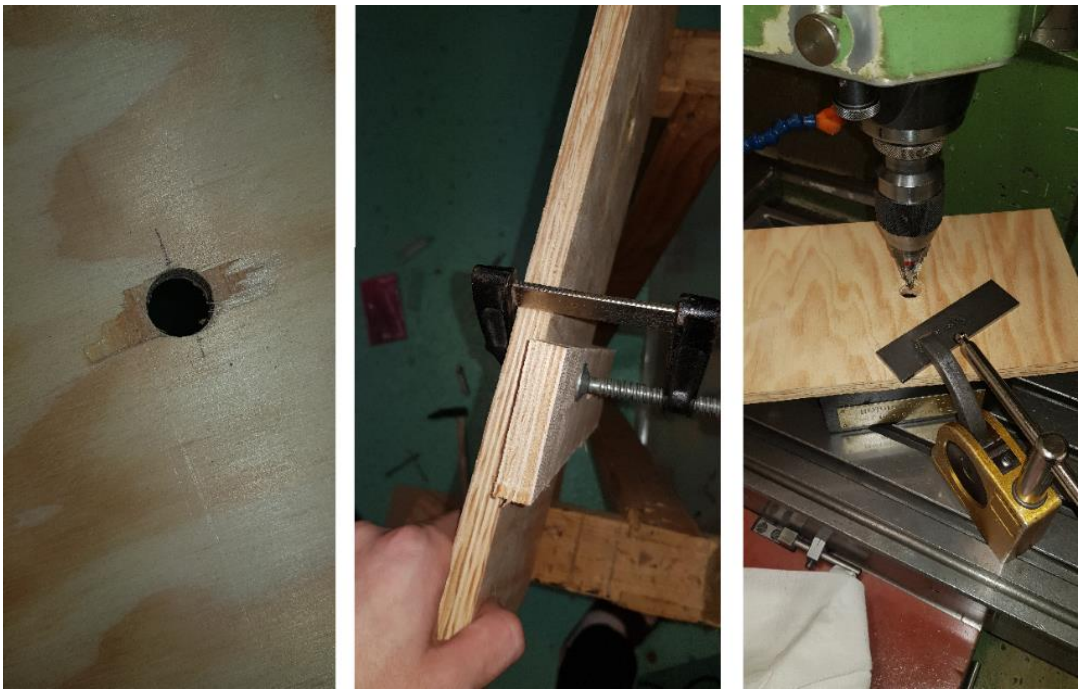


Abbildung 28, Vorgehen bei den Bohrungen

Nach den Problemen, die ich mit dem Holz hatte, würde ich das nächste Mal lieber Vollholzplatten anstelle von geleimten Platten nehmen. Sie kosten mehr, sind aber bei spanenden Arbeiten standhafter.

Nachdem ich alle Ausschnitte soweit zugeschnitten und gestrichen hatte, schnitt ich noch die Holzlatten auf ihre Masse zu, damit ich im nächsten Schritt den Automaten zusammenschrauben konnte. Auch mit den Schrauben begann das Holz auszubrechen. Obwohl die Mängel nicht mehr in dem vorigen Ausmass auftraten, habe ich an den meisten Stellen das Holz noch einmal zusammengeklebt und mit einem Spanner zusammengepresst. Somit ist die Stabilität mehr gewährleistet. Ich habe den Spielautomaten in zwei Teile aufgeteilt, bevor ich ihn komplett zusammengeschraubt habe. Dadurch wurde die Montage der Bedienelemente und des Monitors erleichtert. Die Bedienelemente wurden angeschraubt und den Monitor befestigte ich mit einer Holzlatte am Gestell. Um ein Hin- und Herrücken zu vermeiden, habe ich ihn zusätzlich mit Heissleim befestigt.

Abbildung 29, Halterung Monitor

Mit der Montage des Monitors war es nun soweit die beiden Teile des Automaten zusammenzufügen. Dies war der letzte Schritt des Aufbaus. Probleme gab es ausser der ungeeignet Wahl des Holzes nicht. Ich habe versucht, möglichst genau zu arbeiten, damit beim Zusammenfügen sämtlicher Teile alles passte und nicht anders gelöst werden musste.

Abbildung 30, Teilstücke des Automat

4.3 Verkabelung und Installation

Nach der Fertigstellung des Gestells, habe ich den Raspberry Pi, den Ventilator und die USB-Schnittstelle für die Bedienung eingebaut. Diese drei Komponenten habe ich direkt auf der Bodenplatte des Gestells befestigt.

Für meinen Spielautomaten habe ich einen Raspberry Pi 3 B+ verwendet. Zu der Zeit als ich ihn bestellt habe, war es der neuste verfügbare. Ich habe diesen ausgewählt, weil er am leistungsstärksten ist und somit sicher auch in der Lage war mein Spiel auszuführen. Den Ventilator habe ich aus einem alten Computergehäuse ausgebaut und an ein Netzteil verbunden. Auch wenn der Raspberry Pi auch ohne Kühlung in der Lage sein sollte für längere Zeit in Betrieb zu sein, wollte ich kein Risiko eingehen und habe mich für eine Kühlung entschieden. Ich habe auch Messungen durchgeführt und diese ergaben, dass der Raspberry Pi mit Ventilator 5° Celsius kühler bleibt, als ohne Kühlung.

Die Stromversorgung verläuft über eine Mehrfach-Steckerleiste. Ich habe bewusst das Kabel nicht durch eine Kabelverschraubung nach aussen geführt, damit man das Kabel zum Transport im Inneren des Spielautomaten aufbewahren kann. Alle anderen Kabel, die im Automaten vorhanden sind, sind direkt am Raspberry Pi mit den anderen Komponenten verbunden.

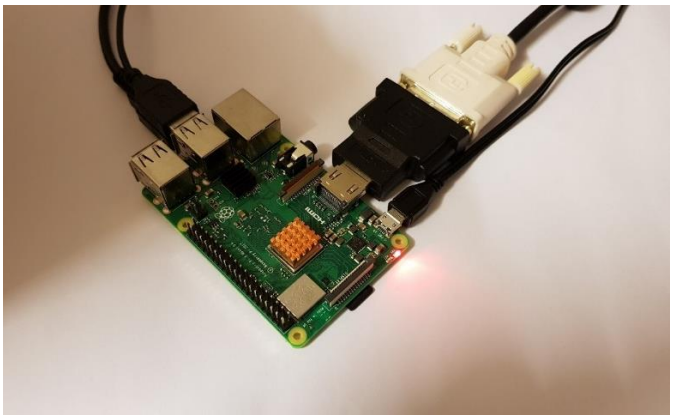


Abbildung 31, Angeschlossener Raspberry Pi

Das Einschalten des Automaten erfolgt durch einstecken des Hauptkabel an eine Stromversorgungsquelle. Das Spiel habe ich auf den Raspberry Pi geladen und ihm gesagt, dass er es direkt beim Aufstarten ausführen soll. Teilweise musste ich den Raspberry Pi mehrmals einschalten, weil er nicht hochfahren wollte. Eine Lösung bzw. eine Ursache für dieses Problem habe ich nicht gefunden. Nichtsdestotrotz lasse ich mich nicht davon stören, solange der Rechner schliesslich startet.

5 Zeitplanung

Bei der Zeitplanung habe ich darauf geachtet, dass ich zuerst die Arbeitsschritte erledige, bei denen ich am wenigsten Vorwissen aufweise. Zusätzlich habe ich noch eine Soll- und Ist-Zeit Tabelle geführt, um zu kontrollieren, ob ich im Zeitplan liege.

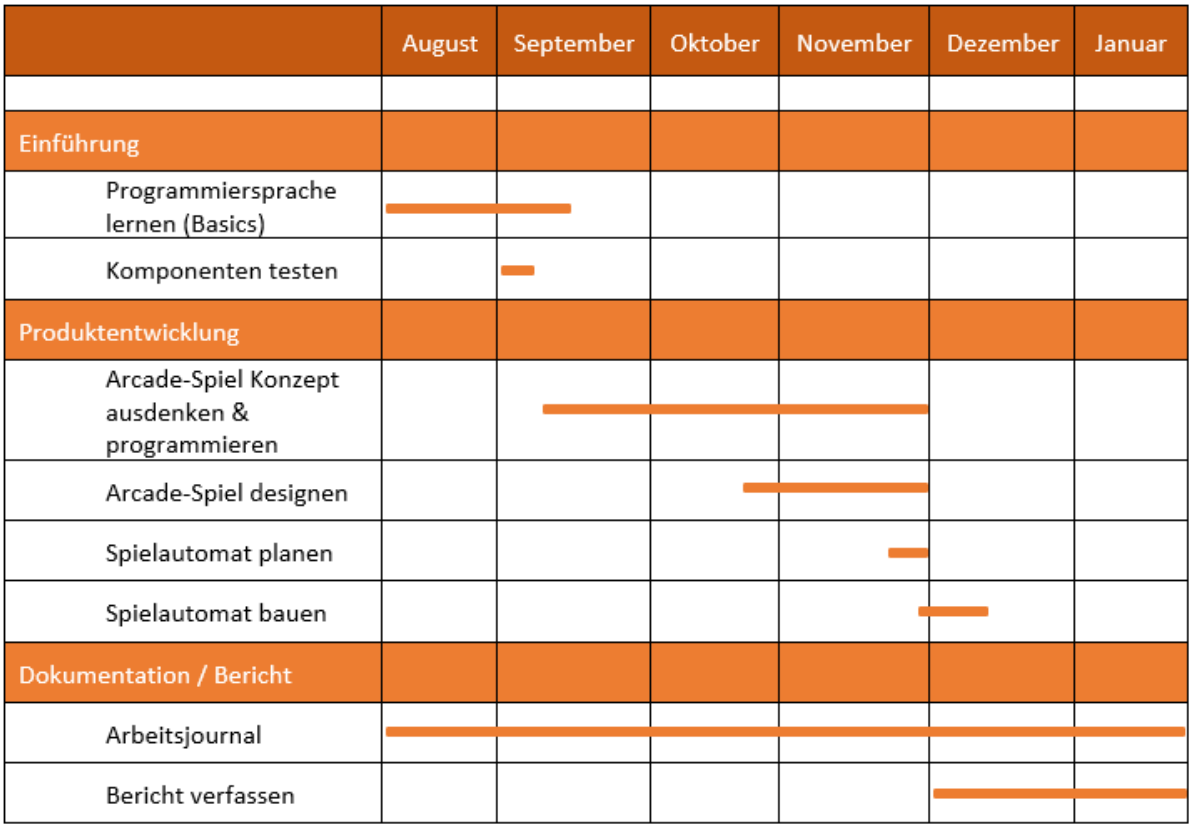


Abbildung 32, Zeitplan mit Deadlines

Arbeitsschritte	Sollzeit [h]	Istzeit [h]
Auswahl der Programmiersprache	2.00	~1.00
Programmiersprache Basics lernen	6.00	4.00
Auswahl Komponenten Spielautomat	1.00	1.00
Komponenten aufsetzen & testen	1.00	2.00
Einführung Total:	10.00	8.00
Spielkonzept entwickeln	1.00	2.50
Spielgrundstein programmieren	6.00	9.00
Spielfunktionen hinzufügen	20.00	28.50
Spielcode Optimierung	10.00	8.00
Spiel designen/zeichnen	10.00	17.50
Sounddesign erstellen	6.00	3.00
Spielautomaten planen	3.00	4.00
Spielautomat bauen	6.00	9.00
Spielautomat verkabeln	2.00	3.00
Produktentwicklung Total:	64.00	84.50

Abbildung 33, Soll- und Ist-Zeit Tabelle

6 Schlusswort

Die Arbeit an diesem Projekt war sehr herausfordernd und trotzdem hat es mir Freude bereitet. Ich habe dieses Thema gewählt, weil ich einen Einblick in die Welt der Informatik wollte und diesen habe ich auch bekommen. Ich habe gelernt, selbst ein Spiel zu entwickeln, programmieren und zusätzlich noch in Form eines Spielautomaten auszuführen.

Die grösste Herausforderung der Arbeit, war das Verfassen des Quellcodes für mein eigenes Spiel. Das war ein Gebiet, auf dem ich mich noch nie gross bewegt habe. Ich habe mit Hilfe von Anleitungen mir selbst das Verständnis dafür beigebracht. Das Erstellen des Videospieles hat mich sehr interessiert, was es mir erleichtert hat, solange an diesem Projekt zu Arbeiten. Auch wenn ich teilweise mehrere Stunden am selben Tag damit verbracht habe, am Computer Zeile an Zeile zu schreiben und Probleme zu lösen, hat es mich nicht aus der Fassung gebracht. Es war eher das Gegenteil. Je länger ich daran arbeitete, desto schwieriger war es für mich damit zu stoppen.

Auch die Konstruktion war kein leichtes für mich, da ich aber in meinem Beruf handwerklich tätig bin, fiel diese Arbeit wesentlich einfacher aus. Ich habe die Erfahrung, die ich aus dem Betrieb habe, angewendet und einen voll funktionsfähigen Spielautomaten damit gebaut. Es gibt Punkte, die ich verbessern beim nächsten Mal verbessern würde und bereits auch schon genannt habe, aber ich bin mit meiner Leistung zufrieden. Es war ja schliesslich das erste Mal, dass ich etwas in dieser Art hergestellt habe.

Mit meinem Endprodukt des Arcade-Spielautomaten bin ich sehr glücklich. Zu Beginn des Projektes hätte ich nie gedacht, dass dieser so aussehen würde. Ich habe damit meine Erwartungen übertroffen. Zusätzlich habe ich mit der Arbeit ein neues Hobby mit dem Programmieren gefunden. Ich werde bestimmt noch einige weitere Projekte in diese Richtung erarbeiten und es hat meine Wahl des Studienganges für nach der Berufsmatura weiter gestärkt. Dank dieser Arbeit bin ich mir sicher, dass ich auf den Bereich der Informatik wechseln will.

Quellenverzeichnis

BFXR. (-). Von https://www.bfxr.net/ abgerufen

Dev Insider. (10. Mai 2017). Von https://www.dev-insider.de/was-ist-eine-ide-a-600703/ abgerufen

Ionos. (23. Januar 2018). Von https://www.ionos.de/digitalguide/websites/web-entwicklung/quellcode/ abgerufen

Pygame. (-). Von https://www.pygame.org/ abgerufen

Raspberry Pi. (-). Von https://www.raspberrypi.org/ abgerufen

SEO-Analyse. (kein Datum). Von https://www.seo-analyse.com/seo-lexikon/q/quellcode/ abgerufen

The Verge. (16. Januar 2013). Von https://www.theverge.com/2013/1/16/3740422/the-life-and-death-of-the-american-arcade-for-amusement-only abgerufen

YouTube. (2017). Von Tech With Tim: https://www.youtube.com/watch?v=i6xMBig-pP4&list=PLzMcBGfZo4-lp3jAExUCewBfMx3UZFkh5&index=1 abgerufen

Abbildungsverzeichnis

Anhang

Quellcode

```
# Import libraries
import pygame as py
import random as rn
import pickle as pk
from os import path

# Declare path from image and sound folder
img_dir = path.join(path.dirname(__file__), 'img')
snd_dir = path.join(path.dirname(__file__), 'snd')

# Game settings
width = 1024
height = 768
fps = 60
game_speed = 2
player_speed = 7
bullet_speed = 7
player_speed1 = 7
player_speed2 = 6
player_speed3 = 5
bullet_speed1 = 7
bullet_speed2 = 6
bullet_speed3 = 5

# Define colors
white = (255,255,255)
black = (0,0,0)
red = (255,0,0)
green = (3,173,76)
blue = (0,0,255)

# Load Highscore
try:
    with open('highscore.dat', 'rb') as file:
        highscore = pk.load(file)
except:
    highscore = 0

# Initialize pygame and create window
py.mixer.pre_init(44100, -16, 2, 2048)
py.mixer.init()
py.init()
screen = py.display.set_mode((width, height))
py.display.set_caption('Land Runner')
py.mouse.set_visible(0)
clock = py.time.Clock()

# Definition for drawing text
def draw_text(surf, text, size, x, y, color):
    font = py.font.Font(path.join(path.dirname(__file__), 'galaxy-monkey.regu-
lar.ttf'), size)
    text_surface = font.render(text, True, color)
    text_rect = text_surface.get_rect()
    text_rect.midbottom = (x, y)
    surf.blit(text_surface, text_rect)
```

```
# Definition for spawning rocks
def newmob():
    m = mob()
    all_sprites.add(m)
    mobs.add(m)

# Definition for spawning enemies
def newenemy():
    e = enemy()
    all_sprites.add(e)
    enemies.add(e)

# Definition for start screen
def start_screen():
    craft = startscreen_ship()
    startscreen_sprites.add(craft)
    lastupdate = py.time.get_ticks()
    waiting = True
    joystick = py.joystick.Joystick(0)
    joystick.init()
    while waiting:
        button = joystick.get_button(0)
        key = py.key.get_pressed()
        nowupdate = py.time.get_ticks()
        clock.tick(fps)
        BackGround.update(screen)
        startscreen_sprites.update()
        startscreen_sprites.draw(screen)
        draw_text(screen, str(highscore), 72, width/2, height/2, white)
        for event in py.event.get():
            if event.type == py.QUIT or key[py.K_ESCAPE]:
                py.quit()
            if button:
                if nowupdate - lastupdate > 2000:
                    craft.kill()
                    waiting = False
        py.display.flip()

def gameoverscreen():
    gameover = True
    lastcheck = py.time.get_ticks()
    while gameover:
        nowcheck = py.time.get_ticks()
        BackGround.update(screen)
        draw_text(screen, 'Score', 72, width/2, height/2 -80, white)
        draw_text(screen, str(score), 72, width/2, height/2, white)
        if nowcheck - lastcheck > 3000:
            gameover = False
        py.display.flip()


# Scrolling Background
class background(py.sprite.Sprite):
    def __init__(self):
        py.sprite.Sprite.__init__(self)
        self.image = bg_img
        self.rect = self.image.get_rect()
        self.y1 = 0
        self.y2 = height * -1

    def update(self, screen):
```

```
        screen.blit(self.image, (0, self.y1))
        screen.blit(self.image, (0, self.y2))
        self.y1 += game_speed
        self.y2 += game_speed

        if self.y1 > height:
            self.y1 = self.y2 - height
        if self.y2 > height:
            self.y2 = self.y1 - height

# Creating startscreen player
class startscreen_ship(py.sprite.Sprite):
    def __init__(self):
        py.sprite.Sprite.__init__(self)
        self.image = start_img[0]
        self.image.set_colorkey(black)
        #self.image.fill(green)
        self.rect = self.image.get_rect()
        self.radius = 28
        self.rect.centerx = width / 2
        self.rect.bottom = height
        self.last = py.time.get_ticks()

    def update(self):
        now = py.time.get_ticks()
        if now - self.last > 200:
            self.image = start_img[1]
            self.image.set_colorkey(black)
        if now - self.last > 400:
            self.image = start_img[0]
            self.image.set_colorkey(black)
            self.last = now

# Creating player sprite
class player(py.sprite.Sprite):
    def __init__(self):
        # Built in sprite init function
        py.sprite.Sprite.__init__(self)
        self.image = player_img[0]
        self.image.set_colorkey(black)
        #self.image.fill(green)
        self.rect = self.image.get_rect()
        self.radius = 28
        #py.draw.circle(self.image, red, self.rect.center, self.radius)
        self.rect.centerx = width / 2
        self.rect.bottom = height - 40
        self.speed_x = 0
        self.speed_y = 0
        self.last = py.time.get_ticks()
        self.last_bullet = py.time.get_ticks()

    def animation(self):
        now = py.time.get_ticks()
        if now - self.last > 200:
            self.image = player_img[1]
            self.image.set_colorkey(black)
        if now - self.last > 400:
            self.image = player_img[0]
            self.image.set_colorkey(black)
            self.last = now
```

```
    # Bewegungen die der Spieler machen kann
    def update(self):
        self.animation()
        self.speed_x = 0
        self.speed_y = 0
        key = py.key.get_pressed()
        joystick = py.joystick.Joystick(0)
        joystick.init()
        updown = joystick.get_axis(1)
        rightleft = joystick.get_axis(0)

        # Rechts links steuerung und Border
        if rightleft <= -0.1 or key[py.K_LEFT]:
            self.speed_x = player_speed *-1
        if rightleft >= 0.1 or key[py.K_RIGHT]:
            self.speed_x = player_speed
        self.rect.x += self.speed_x
        if self.rect.right > width -200:
            self.rect.right = width -200
        if self.rect.left < 200:
            self.rect.left = 200
        # Hoch runter steuerung und Border
        if updown <= -0.1 or key[py.K_UP]:
            self.speed_y = -1 * player_speed
        if updown >= 0.1 or key[py.K_DOWN]:
            self.speed_y = player_speed
        self.rect.y += self.speed_y
        if self.rect.top < 40:
            self.rect.top = 40
        if self.rect.bottom > height - 10:
            self.rect.bottom = height - 10

    def shoot(self):
        bullet_now = py.time.get_ticks()
        if bullet_now - self.last_bullet > 300:
            bullet = Bullet(self.rect.centerx, self.rect.top + 20)
            all_sprites.add(bullet)
            bullets.add(bullet)  # add bullet to a group, so we can add collision
            self.last_bullet = bullet_now
            shoot_sound.play()

# Creating enemy sprite
class enemy(py.sprite.Sprite):
    def __init__(self):
        py.sprite.Sprite.__init__(self)
        self.image = enemyship_img[0]
        self.image.set_colorkey(black)
        self.rect = self.image.get_rect()
        self.radius = 28
        #py.draw.circle(self.image, red, self.rect.center, self.radius)
        self.rect.x = rn.randrange(200, width - 200 - self.rect.width)
        self.rect.y = rn.randrange(-500, -300)
        self.speed_y = game_speed + 1
        self.last = py.time.get_ticks()
        self.lastshot = py.time.get_ticks()
        self.shootcount = 0
        self.shootplace = rn.randrange(0, height/4)
        self.shoottimer = rn.randrange(300, 1000)
        self.shotallow = rn.randrange(1,3)
        self.speed_x = rn.randrange(-1, 1)
```

```
    def animation(self):
        now = py.time.get_ticks()
        if now - self.last > 200:
            self.image = enemyship_img[1]
            self.image.set_colorkey(black)
        if now - self.last > 400:
            self.image = enemyship_img[0]
            self.image.set_colorkey(black)
            self.last = now

    def update(self):
        self.animation()
        self.movex()
        self.speed_y = game_speed + 1
        self.rect.y += self.speed_y
        if self.rect.y > self.shootplace:
            self.shoot()

    def shoot(self):
        ebulletnow = py.time.get_ticks()
        if self.shootcount < self.shotallow:
            if ebulletnow - self.lastshot > self.shoottimer:
                enemybullet = Enemybullet(self.rect.centerx, self.rect.bottom)
                all_sprites.add(enemybullet)
                enemybullets.add(enemybullet)  # add bullet to a group, so we can
add collision
                shoot_sound.play()
                self.shootcount += 1
                self.lastshot = ebulletnow

    def movex(self):
        if score >= 80:
            self.rect.x += self.speed_x
            if self.rect.left < 200 or self.rect.right > width - 200:
                self.speed_x = self.speed_x * -1

# Creating rock sprite
class mob(py.sprite.Sprite):
    def __init__(self):
        py.sprite.Sprite.__init__(self)
        self.image = enemy_img
        self.image.set_colorkey(black)
        #self.image.fill(red)
        self.rect = self.image.get_rect()
        self.radius = 25
        #py.draw.circle(self.image, red, self.rect.center, self.radius)
        self.rect.x = rn.randrange(200, width - 200 - self.rect.width)
        self.rect.y = rn.randrange(-2000, -400)
        self.speed_y = game_speed

    def update(self):
        self.speed_y = game_speed
        self.rect.y += self.speed_y
        #self.rect.x += self.speed_x
        #if self.rect.left < 82 or self.rect.right > width - 82:
            #self.speed_x = self.speed_x * -1
        #if self.rect.top > height + 10:
            #self.rect.x = rn.randrange(82, width - 82 - self.rect.width)
            #self.rect.y = rn.randrange(-3000, -200)
            #self.speed_y = game_speed
```

```
# Creating bullet sprite
class Bullet(py.sprite.Sprite):
    def __init__(self, x, y):
        py.sprite.Sprite.__init__(self)
        self.image = bullet_img
        self.image.set_colorkey(black)
        #self.image.fill(white)
        self.rect = self.image.get_rect()
        self.radius = 8
        #py.draw.circle(self.image, red, self.rect.center, self.radius)
        self.rect.bottom = y
        self.rect.centerx = x
        self.speed_y = -1 * bullet_speed

    def update(self):
        self.rect.y += self.speed_y
        # remove if it moves off the top of the screen
        if self.rect.bottom < 0:
            self.kill()

# Creating enemybullet sprite
class Enemybullet(py.sprite.Sprite):
    def __init__(self, x, y):
        py.sprite.Sprite.__init__(self)
        self.image = enemybullet_img
        self.image.set_colorkey(black)
        self.rect = self.image.get_rect()
        self.radius = 8
        self.rect.top = y
        self.rect.centerx = x
        self.speed_y = game_speed + bullet_speed

    def update(self):
        self.rect.y += self.speed_y
        if self.rect.top > height:
            self.kill()

# Creating explosion sprite
class explosion(py.sprite.Sprite):
    def __init__(self, center, size):
        py.sprite.Sprite.__init__(self)
        self.size = size
        self.image = explosion_animation[self.size][0]
        self.rect = self.image.get_rect()
        self.rect.center = center
        self.frame = -1
        self.last_update = py.time.get_ticks()
        self.frame_rate = 30
        self.speed_y = game_speed

    def update(self):
        self.rect.y += game_speed
        now = py.time.get_ticks()
        if now - self.last_update > self.frame_rate:
            self.last_update = now
            self.frame += 1
            if self.frame == len(explosion_animation[self.size]):
                self.kill()
            else:
                center = self.rect.center
```

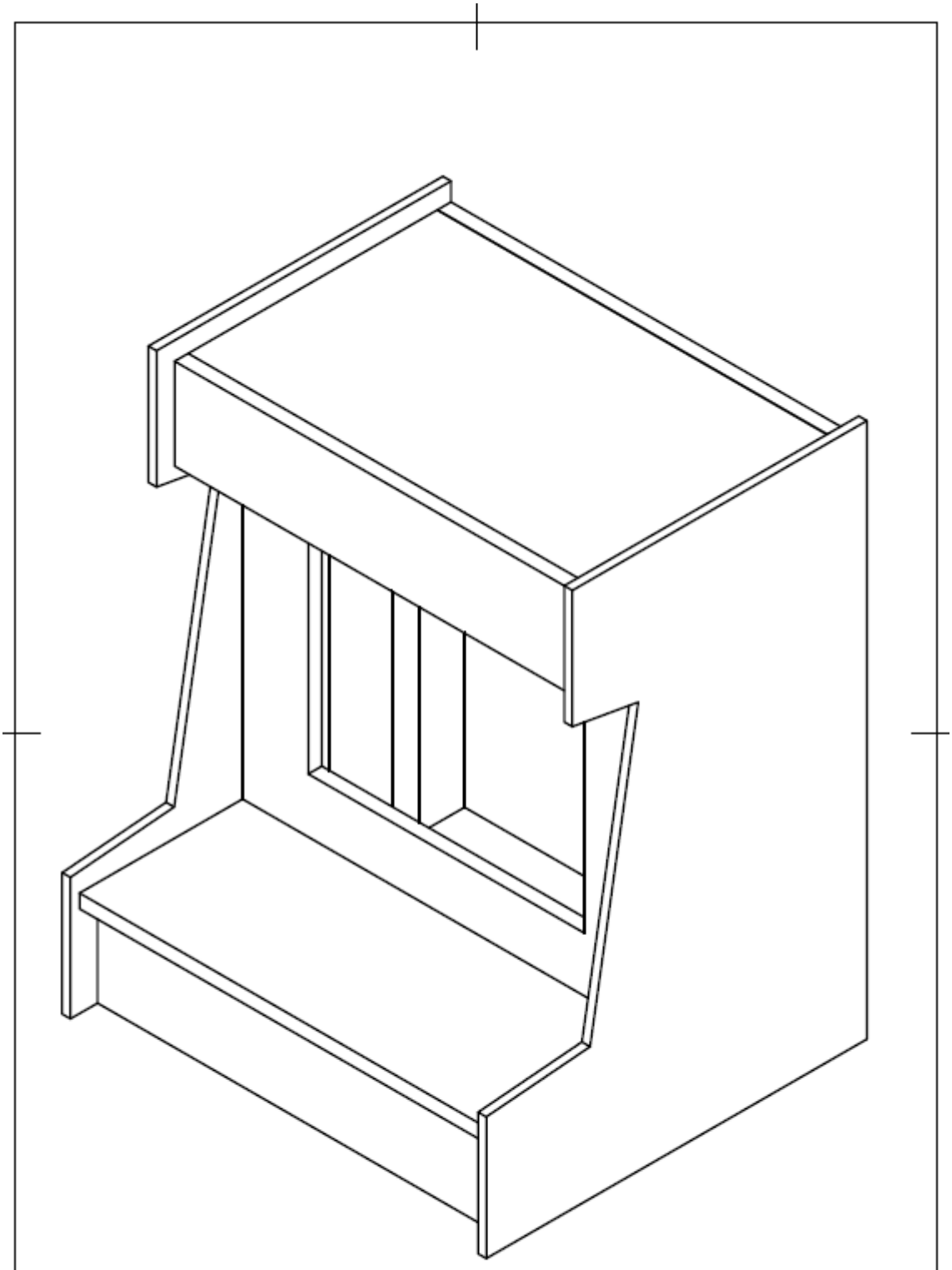
```
                self.image = explosion_animation[self.size][self.frame]
                self.rect = self.image.get_rect()
                self.rect.center = center

# Creating border sprite
class border(py.sprite.Sprite):
    def __init__(self):
        py.sprite.Sprite.__init__(self)
        self.image = py.Surface((1024, 10))
        self.rect = self.image.get_rect()
        self.rect.x = 0
        self.rect.y = height + 100

# Load all game graphics
start_img1 = py.image.load(path.join(img_dir, "startscreen1.png")).convert()
start_img2 = py.image.load(path.join(img_dir, "startscreen2.png")).convert()
start_img = [start_img1, start_img2]
bg_img = py.image.load(path.join(img_dir, "bg.png")).convert()
player_img1 = py.transform.scale(py.image.load(path.join(img_dir,
"player1.png")).convert(), (72, 92))
player_img2 = py.transform.scale(py.image.load(path.join(img_dir,
"player2.png")).convert(), (72, 92))
player_img = [player_img1, player_img2]
bullet_img = py.transform.scale(py.image.load(path.join(img_dir, "bul-
let.png")).convert(), (20,30))
enemybullet_img = py.transform.scale(py.image.load(path.join(img_dir, "bulle-
tenemy.png")).convert(), (20,30))
enemy_img = py.transform.scale(py.image.load(path.join(img_dir,
"enemy.png")).convert(), (70,80))
enemyship_img1 = py.transform.scale(py.image.load(path.join(img_dir,
"enemyship1.png")).convert(), (72, 92))
enemyship_img2 = py.transform.scale(py.image.load(path.join(img_dir,
"enemyship2.png")).convert(), (72, 92))
enemyship_img = [enemyship_img1, enemyship_img2]
explosion_animation = {}
explosion_animation['lg'] = []
explosion_animation['sm'] = []
for i in range(8):
    filename = 'explosion-{}.png'.format(i)
    img = py.image.load(path.join(img_dir, filename)).convert()
    img.set_colorkey(black)
    img_lg = py.transform.scale(img, (60,60))
    explosion_animation['lg'].append(img_lg)
    img_sm = py.transform.scale(img, (25,25))
    explosion_animation['sm'].append(img_sm)

# Load all game sounds
shoot_sound = py.mixer.Sound(path.join(snd_dir, "lasershoot.wav"))
shoot_sound.set_volume(0.25)
explosion_enemy_sound = py.mixer.Sound(path.join(snd_dir, "explosionenemy.wav"))
explosion_enemy_sound.set_volume(0.25)
explosion_player_sound = py.mixer.Sound(path.join(snd_dir, "explosionplayer.wav"))
explosion_player_sound.set_volume(0.25)
explosion_bullet_sound = py.mixer.Sound(path.join(snd_dir, "explosionbullet.wav"))
explosion_bullet_sound.set_volume(0.05)
py.mixer.music.load(path.join(snd_dir, "theme.ogg"))
py.mixer.music.set_volume(0.15)

# Start scrolling Background
BackGround = background()
```

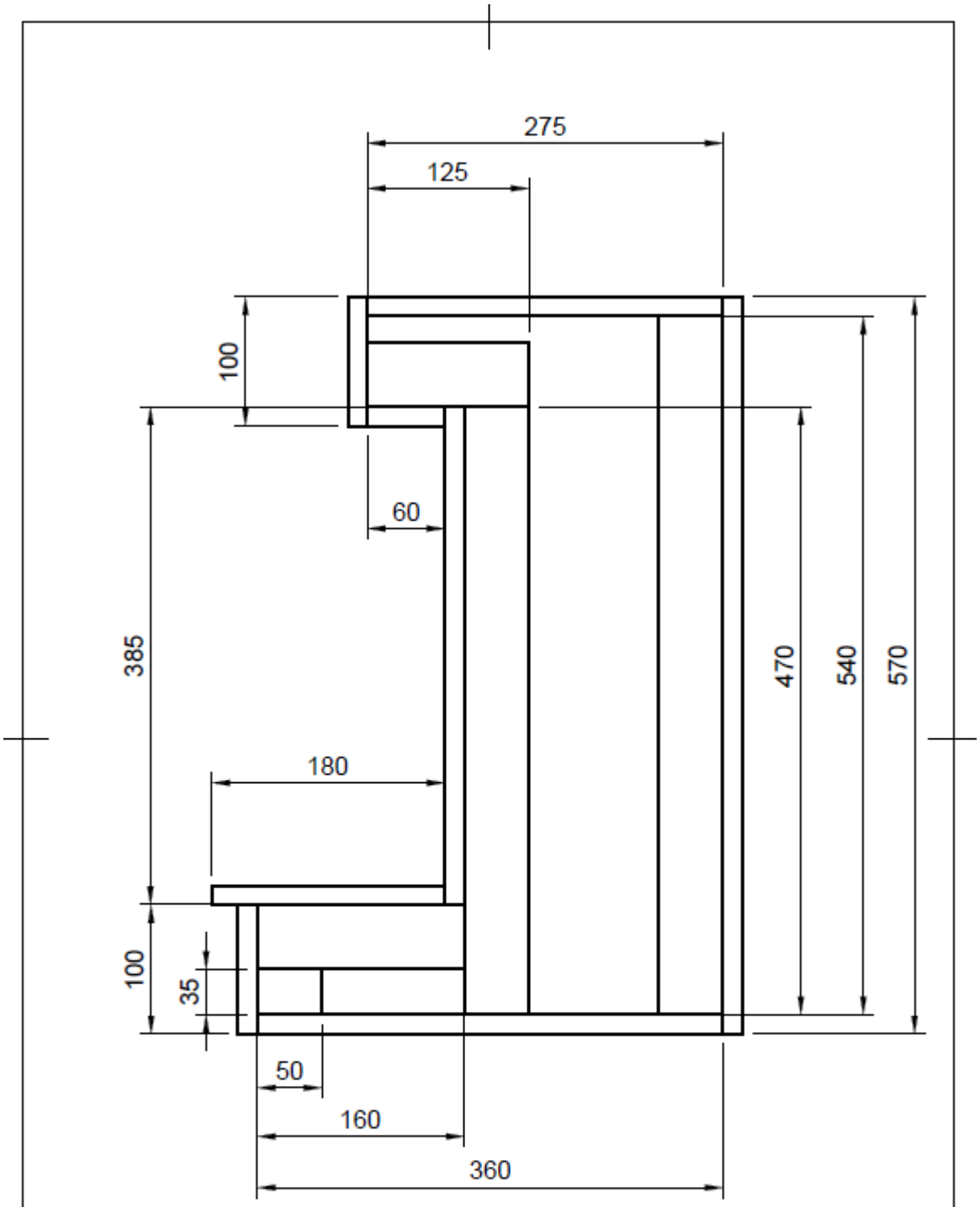
```
# Start playing game music
py.mixer.music.play(loops=-1)

# Groups
startscreen_sprites = py.sprite.Group()

#----------Game Loop----------#
game_over = True
run = True
while run:
    if game_over:
        start_screen()
        game_over = False
        # All the sprites
        all_sprites = py.sprite.Group()
        enemies = py.sprite.Group()
        bullets = py.sprite.Group()
        enemybullets = py.sprite.Group()
        Player = player()
        Border = border()
        mobs = py.sprite.Group()
        all_sprites.add(Player)
        all_sprites.add(Border)
        for i in range(6):
            newmob()
        for i in range(1):
            newenemy()

        #add score
        score = 0
        counter = 0
    clock.tick(fps)
    # Game speed

    if score >= 5:#5
        if counter == 0:
            fps = 80
            player_speed = player_speed2
            bullet_speed = bullet_speed2
            counter = 1
    if score >= 10:#15
        if counter == 1:
            newenemy()
            counter = 2
    if score >= 20:#25
        if counter == 2:
            fps = 100
            player_speed = player_speed3
            bullet_speed = bullet_speed3
            counter = 3
    if score >= 30:#40
        if counter == 3:
            fps = 60
            game_speed = 3
            player_speed = player_speed1
            bullet_speed = bullet_speed1
            counter = 4
    if score >= 50:#55
        if counter == 4:
            fps = 80
            player_speed = player_speed2
```

```
            bullet_speed = bullet_speed2
    if score >= 70:#80
        fps = 100
        bullet_speed = bullet_speed3
        player_speed = player_speed3


    #initialize variables
    key = py.key.get_pressed()
    # Keep loop running at the right speed

    # Process input
    for event in py.event.get():
        #check for closing window
        if event.type == py.QUIT or key[py.K_ESCAPE]:
            run = False
        #if event.type == py.KEYDOWN:
            #if event.key == py.K_SPACE:
                #Player.shoot()
        if event.type == py.JOYBUTTONDOWN:
            if event.button == 0:
                Player.shoot()

    # Update
    all_sprites.update()

    # Check to see if a bullet hits enemies
    bullethits = py.sprite.groupcollide(enemies, bullets, True, True,
py.sprite.collide_circle)
    for hit in bullethits:
        score += 1
        explosion_enemy_sound.play()
        expl = explosion(hit.rect.center, 'lg')
        all_sprites.add(expl)
        newenemy()
    # Check to see if a bullet hits rock
    rockhits = py.sprite.groupcollide(bullets, mobs, True, False, py.sprite.col-
lide_circle)
    for hit in rockhits:
        explosion_bullet_sound.play()
        explsm = explosion(hit.rect.center, 'sm')
        all_sprites.add(explsm)
    # Check to see if a mob hits the border
    mobdodge = py.sprite.spritecollide(Border, mobs, True)
    if mobdodge:
        newmob()
    enemydodge = py.sprite.spritecollide(Border, enemies, True)
    if enemydodge:
        newenemy()
    # Check to see if a mob hit the player
    hits = py.sprite.spritecollide(Player, mobs, True, py.sprite.collide_circle)
    if hits:
        explosion_player_sound.play()
        death_explosion = explosion(Player.rect.center, 'lg')
        all_sprites.add(death_explosion)
        Player.kill()
    hits2 = py.sprite.spritecollide(Player, enemies, True, py.sprite.collide_cir-
cle)
    if hits2:
        explosion_player_sound.play()
        death_explosion = explosion(Player.rect.center, 'lg')
```

```
            all_sprites.add(death_explosion)
            Player.kill()
        hits3 = py.sprite.spritecollide(Player, enemybullets, True, py.sprite.col-
lide_circle)
        if hits3:
            explosion_player_sound.play()
            death_explosion = explosion(Player.rect.center, 'lg')
            all_sprites.add(death_explosion)
            Player.kill()


        if not Player.alive() and not death_explosion.alive():
            if score > highscore:
                highscore = score
            with open('highscore.dat', 'wb') as file:
                pk.dump(highscore, file)
            game_speed = 2
            fps = 60
            bullet_speed = 7
            player_speed = 7
            game_over = True
            gameoverscreen()


        # Draw / Render
        screen.fill(black)
        BackGround.update(screen)
        draw_text(screen, str(score), 80, width / 2, 90, green)
        all_sprites.draw(screen)

        # After drawing everything ,flip the display
        py.display.flip()

py.quit()
```

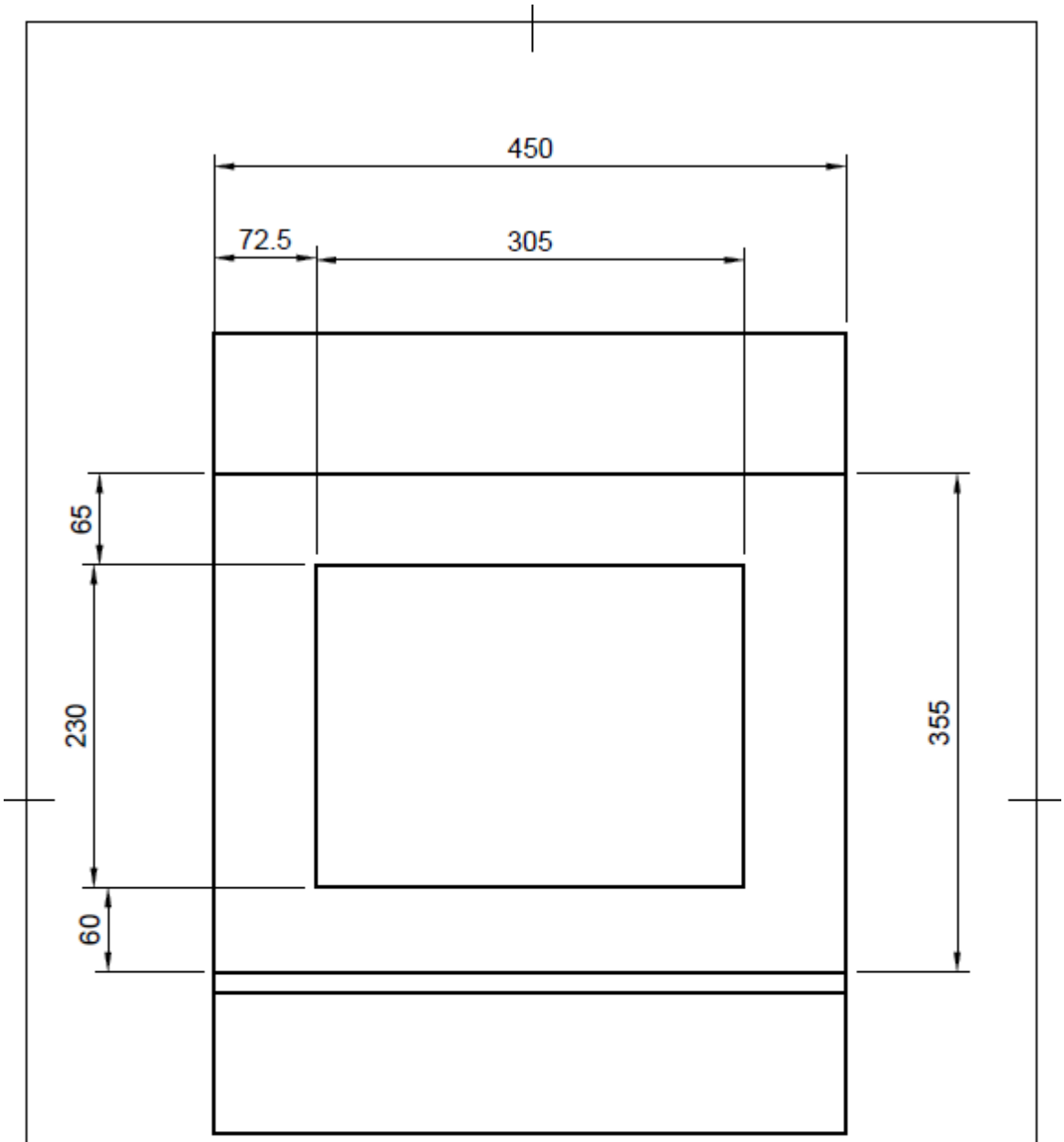
Zeichnungen Konstruktion





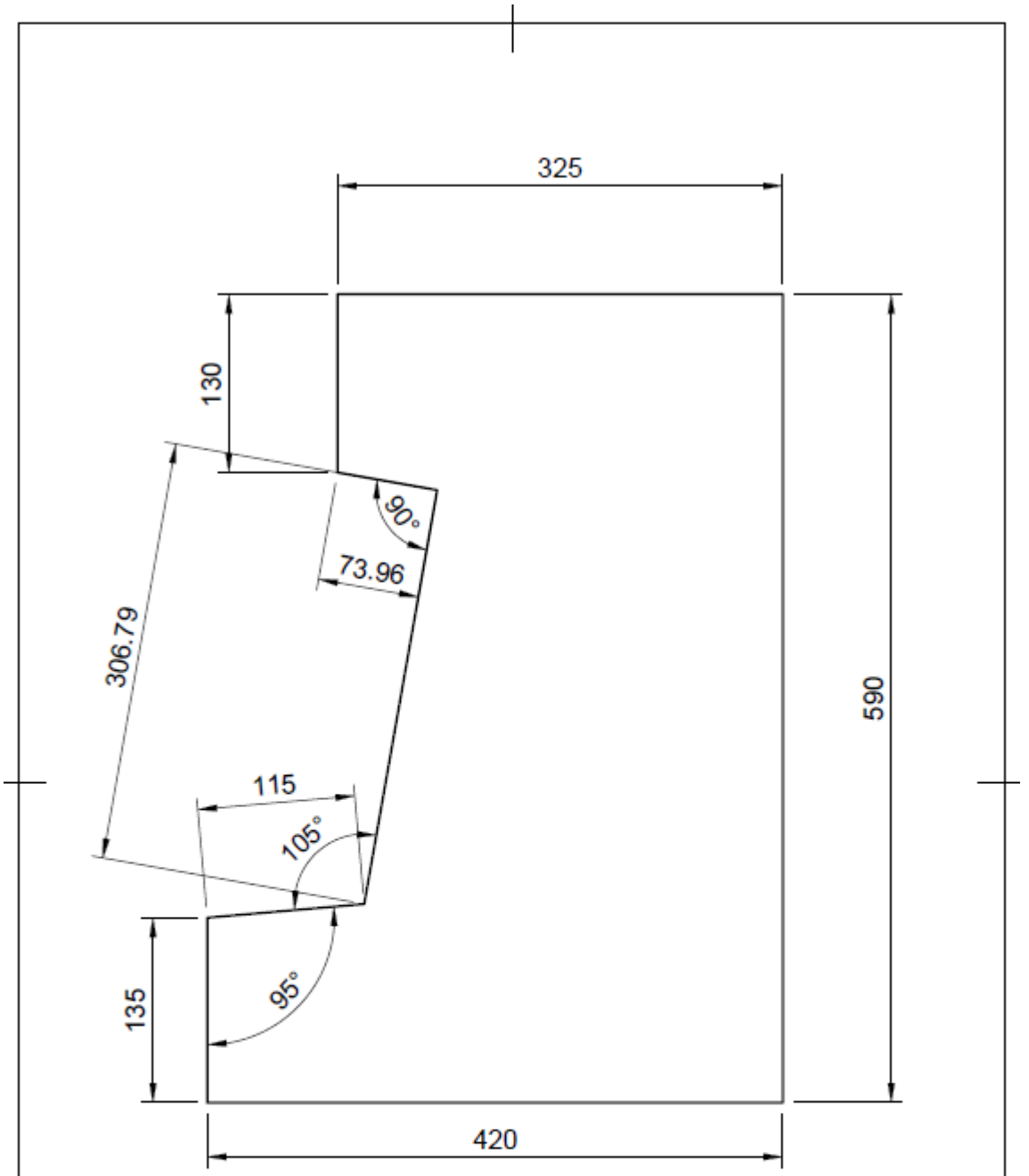
1:4

Gezeichnet von	Titel	Datum	
Marvin Büeler	Gerüst Seite	07.12.2019	2/4



1:4

Gezeichnet von	Titel	Datum	
Marvin Büeler	Gerüst Front	07.12.2019	3/4



1:4

Gezeichnet von	Titel	Datum	
Marvin Büeler	Seitenwand	07.12.2019	4/4

LI-4.3.5.4-01

bbzg	4.3.5 Berufsmaturität **LI-4.3.5.4-01** **Themenfindung / Planung IDPA**	**berufsbildungszentrum**goldau Version 6.0 | Seite 1 von 5
Allgemeine Angaben, Grobstruktur, Fachbereiche, Arbeitsablauf, Rückmeldung		

1. Allgemeine Angaben

Name: Büeler Vorname: Marvin

Beruf: Automatiker Wohnort: Lachen SZ

Oberthema: Bau eines Arcade-Spielautomaten

Zwischenorientierung:

Abgabetermin:

---

2. Grobstruktur

Stichwortartiger Inhalt:

- Entwicklung eines Arcade Spiels
- Das Arcade Spiel auf einen Raspberrypi übertragen
- Bau des Spielautomaten aus Holz
- Einbau der Elektrokomponenten (Display, Taster und Joystick)

Als Titel meiner Arbeit wähle ich:

Arcade-Spielautomat

Vorgesehene Quellen:

-Internet
Etc.

Bearbeitungsangaben	Erstellt am 07.06.16	Vis. CH	Geprüft am 07.06.16	Vis. CH	Freigegeben am 07.06.16	Vis. RK
Dateiname: C:\Users\adrian.bingisser\AppData\Local\Microsoft\Windows\INetCache\Content.Outlook\W1WK8P6H\Themenfindung IDPA Marvin Büeler.doc					Ersetzt Version 5.0 vom 01.01.15	

3. Teilziele der Arbeit:

Durch mein Projekt möchte ich mich an die Materie der Programmiersprachen und allgemein der Informatik annähern. Ich interssiere mich sehr für diese Themen und möchte später auch in diesem Bereich stuudieren. Da ich jedoch noch fast keine Erfahrung mit der Spieleentwicklung habe, werde ich im laufe der Zeit entscheiden, in welcher Art das Spiel sein wird. Das ganze werde ich mit den RaspberryPi kompatiblen Programmen «Python» und «Pygame» realiesieren. Das Gestell des Spielautomaten werde ich selber konstruiren und aus Holz gestalten.

Die Arbeit möchte ich im folgenden Fach präsentieren:
Berufskunde

Weitere Bemerkungen

Ort, Datum, Unterschrift Lernender: Goldau 3.7.19 M. Fuchs

4. Planung des Arbeitsablaufs:

Termine	Arbeitsschritte
20.07.2019	- Planung des Spielautomaten beenden
03.08.2019	- Gestell des Spielautomaten bauen - Elektro-Komponenten einbauen
30.09.2019	- Arcade-Spiel fertigstellen, damit noch genug Zeit für Fehlerbehebungen und Erweiterungen übrig ist.
30.11.2019	- Dokumentation (schriftlicher Teil der Arbeit) beenden

bbzg	4.3.5 Berufsmaturität LI-4.3.5.4-01 Themenfindung IDPA	berufsbildungszentrumgoldau	
		Version 6.0	Seite 5 von 5
Allgemeine Angaben, Grobstruktur, Teilziele, Arbeitsablauf, Rückmeldung			

5. Rückmeldungen der beurteilenden Lehrperson:

Ort, Datum, Unterschrift der beurteilenden Lehrperson :

Goldau, 1.7.2019

LI-4.3.5.4-02

bbzg	4.3.5 Berufsmaturität LI-4.3.5.4-02 Arbeitsjournal / Reflexion IDPA	berufsbildungszentrumgoldau Version 3.1 | Seite 1 von 2
Arbeitsschritte / Reflexionen / Bemerkungen / Erfahrungen		

6. Arbeitsjournal / Reflexion

Blatt Nr. .....

Datum	Ausgeführte Arbeit / Arbeitsschritte / Reflexionen / Bemerkungen / Erfahrungen / Reflexion	Datum/ Visum
20.06.2019	- Allgemeine Recherchen zu Arcade-Spielautomaten gemacht - Weitere Recherchen zur Entwicklung eines Arcade-Spiels und wie dieses gespielt werden kann - Erstmals auf RaspberryPi gestossen (wird sehr oft für Arcade-Projekte verwendet)	
29.06.2019	- Weitere Recherchen über den RaspberryPi gemacht (15min) - Mich über die Engine Python (wurde auf der RaspberryPi Webseite vorgeschlagen) schlau gemacht (30min) - Spieleentwicklungs Erweiterung «Pygame» entdeckt	
03.08.2019	- Start mit einer Youtube-Tutorialreihe für die Spieleentwicklung mit Pygame (30min) - Erste Basics gelernt, wie z.B. einfügen von Grafiken und Bewegungen	
12.08.2019	- Youtube-Tutorialreihe weitergeführt (1.5h) - Gelernt Animationen einzufügen und den Code zu optimieren	
17.08.2019	- Yoututbe-Tutorialreihe beendet (2h) - Gelernt Musik und Soundeffekte einzufügen und weitere Funktionen für die Spielentwicklung	
25.08.2019	- Bauteile für den Arcade-Automaten im Internet gesucht, wie z.B. Monitor,Knöpfe, Joystick usw. (30min) - Masse der Bauteile sind wichtig für den Aufbau, deshalb bestelle ich diese vor der genauen Planung des Automaten - Monitor auf Ricardo ersteigert	
29.08.2019	- Weiter nach Bauteilen im Internet gesucht (30min) - Joystick, Knöpfe und Encoder bestellt - Raspberry Pi 3 bestellt	
30.08.2019	- Besprechung mit Lehrperson A. Bingisser über den Zwischenstand meiner Arbeit	
08.09.2019	- RaspberryPi eingerichtet, SD-Karte mit Laptop konfiguriert, PC kann SD-Karten nicht lesen (2h) - OS auf Raspberry Pi geladen - PyGame auf den Raspberry Pi installiert - Somit kann ich mein Spiel fortlaufend auf dem RaspberryPi testen	
20.09.2019	- Gedanken und Recherchen zum Konzept meines Spiels gemacht (2.5h) - Das Internet durchsucht, welche Art von Arcade-Spielen es überhaupt gibt und abgeschätzt was für mich möglich ist zu programmieren - Für ein Shoot em up Spiel entschieden und erste Skizze erstellt - Zeitplan erstellt (1.5h)	
15.10.2019	- Spiel entwicklung gestartet (3h) - Spiel einstellungen programmiert wie z.B. grösse des Fensters, Farben usw. - Getestet auf RaspberryPi und dem dazugehörigen Monitor, den ich für den Automaten verwenden werde	
16.10.2019	- Erste Funktionen ins Spiel eingebaut (6h) - Spieler und Gegner in Form von Rechtecken eingefügt - Bewegungen der Genger bestimmt - Schwirigkeiten mit der Konfiguration der Spieler-Steureung, damit diese über den Joystick ermöglicht wird - Auf Pygame-Webseite lösung gefunden	
17.10.2019	- Schiess-Funktion für den Spieler eingefügt (5h) - Wieder längere Zeit gebraucht, damit der Schuss über einen Knopf und nicht die Tastatur ausgeführt werden kann - Getestet auf RaspberryPi	
19.10.2019	- Design des Hintergrunds anhand der Skizze gezeichnet und Spielfeld grösse im Script angepasst (6h) - Hintergrund gezeichnet, Endprodukt entspricht aber noch nicht meinen Erwartungen	
01.11.2019	- Hintergrund verbessert (3h) - Hintergund in Spiel einfügen und diesen von Oben nach Unten unendlich lang laufen lassen (4h)	

Bearbeitungsangaben	Erstellt am 26.10.06	Vis. Sti.	Geprüft am 26.10.06	Vis. RK	Freigegeben am 26.10.06	Vis. Ta
Dateiname: C:\Users\Marvin\Desktop\IDPA\IDPA Dateis\LI-4.3.5.4-02 Arbeitsjournal_Reflexion IDPA.doc					Ersetzt Version 3.0 vom 11.10.04	

bbzg	4.3.5 Berufsmaturität LI-4.3.5.4-02 Arbeitsjournal / Reflexion IDPA	berufsbildungszentrumgoldau Version 3.1 | Seite 2 von 2

Arbeitsschritte / Reflexionen / Bemerkungen / Erfahrungen

02.11.2019	- Spielcharakter gezeichnet (1h) - Charakteranimation gezeichnet und diese in Code animiert (4h) - Schüsse und Gegner gezeichnet und Grafik ins Spiel eingefügt (4h) - Hitbox aller Sprites im Spiel angepasst, damit zusammenstoss realistischer wirkt(2h) - Getestet auf RaspberryPi
09.11.2019	- Soundeffekte und Spiel Musik hinzugefügt (3h) - Explosionsanimation gezeichnet und ins Spiel eingefügt (5h) - Score mit bestimmter Schriftart eingefügt (1h) - Getestet auf RaspberryPi
15.11.2019	- Besprechung mit Lehrperson A. Bingisser über den Zwischenstand meiner Arbeit
16.11.2019	- Weiteren Gegner gezeichnet und ins Spiel eingefügt (2h) - Speicherung des Highscores realisiert (2h)
19.11.2019	- Start Screen gezeichnet (10min) - Start Screen programmiert (3h) - Code optimiert für den RaspberryPi (3h) - RaspberryPi konfiguriert, dass das Spiel beim Boot gestartet wird (1h)
20.11.2019	- Redesign von Gegner Objekt und Start Screen (1h) - Einfügen von Game Over Screen (0.5h) - Gegner Schiessfunktion eingefügt (2h) - Steuerung durch Joystick und Knopf optimiert (1h) - Code Optimierung für den Raspberry Pi (2h)
27.11.2019	- Spielautomat im Autodesk Fusion 360 gezeichnet (4h)
30.11.2019	- Material für den Spielautomaten eingekauft und grob zuschneiden lassen (1h)
07.12.2019	- Ausschnitte aus den Holzplatten mit der Stichsäge gesagt und Löcher gebohrt (5h)
14.12.2019	- Einzelteile vom Spielautomaten zusammen montiert, gestrichen und Elektronik eingebaut (7h)

Bearbeitungsangaben	Erstellt am 26.10.06	Vis. Sti.	Geprüft am 26.10.06	Vis. RK	Freigegeben am 26.10.06	Vis. Ta
Dateiname: C:\Users\Marvin\Desktop\IDPA\IDPA Dateis\LI-4.3.5.4-02 Arbeitsjournal_Reflexion IDPA.doc					Ersetzt Version 3.0 vom 11.10.04	