Projektprotokoll Python-Pygame Projekt

von Benjamin Marx

24.11.2023

Heute habe ich mir erstmal eine Projektidee überlegt. Das Spiel soll eine Variante des Klassikers 2048 (https://play2048.co) werden, also programmiere ich mir erstmal die Grundversion des Spiels, um daraus die Variante zu erstellen. Die Grundidee dabei ist es, das Grundprinzip des Spiels, mit den „tiles“ (einzelne Kacheln in dem Gitternetz), die sich allesamt in eine Richtung bewegen, mit besonderen Leveln zu verbinden, sodass eine Art „Puzzlespiel“ entsteht, bei dem man im besten Fall vor jedem Zug genau nachdenken muss. Ich denke dies kann ich erreichen, wenn überall auf dem Feld Gefahren herrschen. Ideen für Funktionen:

- Gegner -> Andere Farbe, können sich auch verdoppeln (Gegner mit Wert 2, auf Gegner mit Wert 2 = Gegner mit Wert 4). Wenn ein Gegner in ein normales Tile geschoben wird, bleibt nur das Größere.
- Alles bewegt sich nur um 1 in eine Richtung, anstatt bis zum Rand -> dadurch könnte es wichtiger sein über jeden Zug genau nachzudenken.
    - Damit ggf. auch Böden, die die Richtung von Tiles, die darauf kommen, ändern
- Wände -> Böden, über die andere Tiles nicht durchkönnen.

Zudem habe ich das gegebene Pygame Template ein wenig angepasst, darunter die Aufteilung der gameloop in Input, logic und draw, als auch die Aufteilung der Datein in `main.py` (Gameloop), `config.py` (Konfiguration, also Variablen, als auch später die Levels) und `Logic.py` (Also die Funktionen) des Spiels

25.11.2023

Heute habe ich angefangen mit der Grundidee von 2048. Dazu habe ich mir überlegt, wie ich das Board am besten umsetzten kann. Ich habe mich entschieden das board immer in einer Matrix zu speichern. Ein 3x3 Board mit einem tile mit dem Wert 2 in der rechten unteren Ecke sähe also so aus:

```
Board= [[0,0,0], [0,0,0], [0,0,2]]
```

Nun habe ich mir überlegt welche Funktionen ich für das Grundspiel brauche (noch nicht die Fensterausgabe, sondern zuerst nur für das Kern-Spiel. Dieses kann ich dann in der konsole Testen.)

- Funktion zum Bewegen des Feldes in eine der 4 Richtungen
- Funktion, um zufällig eine 2 oder 4 in ein leeres Feld zu tun
- Funktion, die schaut, ob man schon gewonnen hat (2048-tile auf dem Feld)
- Funktion, die schaut, ob man noch einen Zug machen kann (sonst hat man verloren)

Heute habe ich mir also überlegt, wie ich die wohl wichtigste Funktion (die move Funktion) umsetzten kann. Da ich nicht für jede der vier Richtungen eine eigenständige Funktion schreiben möchte, schreibe ich einmal eine `move`-Funktion für eine Richtung, und lasse dann ggf. für die anderen Richtungen das board vor und nach der `move`-Funktion drehen. So wäre die Bewegung

nach oben `up()`, aber die Bewegung nach rechts 3* `rotateright()`, `up()` und `rotateright()`.

Die `up()` Funktion funktioniert folgendermaßen:

1. Das board kopieren -> damit ich darin Änderungen machen kann
2. Mache folgendes für jede Zeile nacheinander (von oben):
   - Für jedes Tile in dieser Zeile:
     a. Wenn eine Null über dem Tile liegt (also frei ist):
        i. Bewege das Tile eins nach oben (aufs freie Feld), und wiederhole a, um zu schauen, ob man das Tile noch eins nach oben setzen kann
     b. Wenn ein das Tile über dieser Zeile dasselbe ist wie dies hier, merge beide zusammen.

Schritt 2 wird von oben nach unten ausgeführt, da so sichergestellt werden kann, dass jedes Tile an dem richtigen Platz landet. Wenn es z.B. von unten nach oben bewegt werden würde, würden tiles weiter unten an tiles, die eigentlich nach oben noch Platz zum Bewegen hätten festhängen.

Ein Problem, was ich bei der Implementierung dieser Funktion hatte, ist dass die Kopie `inp_board=board` nur auf das board verweist, es aber nicht wirklich kopiert. Gelöst habe ich das mit einer völlig unabhängigen Kopie, mithilfe von `copy.deepcopy()` (aus der Library `copy`). Geholfen dabei haben mir folgende Artikel:

https://www.informit.com/articles/article.aspx?p=453682&seqNum=4

https://levelup.gitconnected.com/understanding-reference-and-copy-in-python-c681341a0cd8

Eine weitere Sache, die ich noch machen musste, war eine Möglichkeit für das mergen, also dem Zusammenschieben zweier gleicher tiles. Da ich in Punkt b nicht einfach sagen konnte `board[oberes_tile]=board[unteres_tile]*2`(*pseudocode*), da es sonst möglich wäre für andere tiles von unten, mit dem gerade gemergeten tile zu mergen, brauchte ich eine Möglichkeit die zu verdoppelnden tiles zu markieren und später zu verdoppeln. Dies habe ich einfach durch das Negieren gemacht:

```
board[oberes_tile]=board[unteres_tile]*-1
```

um dann später in einer `merge()` Funktion zu verdoppeln:

```
board[oberes_tile]=board[unteres_tile]*-2
```

28.11.2023

Nun, da ich die Funktion zum Bewegen, mitsamt dem mergen habe, habe ich mir ein wenig Gedanken gemacht, welche Status das Spiel braucht. Mit Status meine ich, wann welche Funktion ausgeführt werden soll, bzw. was ins Fenster gemalt werden soll. Dafür habe ich einen Graphen gemacht, denn ich in schön an das Dokument anhängen werde.

Den aktuellen Status möchte ich immer als string in einer Variable speichern, und in der Gameloop sollen je nach aktuellem Wert (Also dem String) der Variable bestimmte Sachen ausgeführt werden.

Nun habe ich schonmal alle Funktionen, die ich laut meinem Graphen brauche, als leere Funktionen aufgeschrieben, und jenach `status` in der Gameloop ausgeführt, auch wenn das aufgrund der leeren Funktionen noch nichts ergab. Die Funktionen:

- `drawUiStart()`
- `buildBoard()` -> erstellt und füllt am Anfang des Spiels das 4x4 Board
- `drawUiIngame()` -> malt die Ui im Spiel
- `drawBoard()` -> malt das Board in das Fenster
- `bewegungMoeglichSpeziell()` -> prüft, ob man in eine Richtung bewegen kann
- `move()` -> bewegen, bassiert auf `up()` und `rotateRight()`
- `createRandom()` -> zufällige tiles einbauen

1.12.2023

Heute habe ich mich um `rotateRight()` gekümmert, damit ich auch Funktionen für das Bewegen in alle 4 Richtungen, und nicht nur `up()` schreiben kann. Den genauen Prozess, wie ich mithilfe von `rotateRight()`, nur eine `up()` Funktion brauche, und diese komplexe Funktion nicht für alle 4 Richtungen wiederholt schreiben muss, habe ich schon oben besprochen.

Die `rotateRight()` Funktion, funktioniert folgendermaßen:

- Kreiert eine neue Liste, mit vertauschter Breite, und Höhe der originalen Liste.
- Nun geht er jedes Element der alten Liste durch, und fügt es an der rotierten Position ein

Diesen Prozess wiederholt sie `anzahl` mal.

2.12.2023

Heute habe ich die Funktionen `buildBoard()` und `createRandom()` erstellt. Dabei gab es eigentlich keine Schwierigkeiten. Die Idee hinter `createRandom()` ist folgende:

- Alle Felder im Board durchgehen
- Koordinaten in Listen rausschreiben, falls es eine Null ist (Also ein freies Feld)
- Zufällige Zahl, 2 oder 4, in zufällige Koordinaten, aus der Koordinatenliste, in das Board schreiben

Das Ganze wiederholt sich dann n-mal, wobei n in der Funktion mitgegeben werden kann, aber standardmäßig auf 1 ist, also wird 1 zufälliges leeres Feld befüllt.

Für die Zufallszahlen verwende ich die Library `random`, mit den Funktionen `randint()`, für einen Zufälligen Integer und `choice()`, für eine zufällige Auswahl.

3.12.2023

Heute habe ich so weit programmiert, dass das gesamte Spiel in der Konsole spielbar ist.

Dafür habe ich die Funktion `BewegungMoeglichGenerell()`, welche wiedergibt, ob aktuell auf dem eine der 4 Bewegungen möglich wäre. Diese benutze ich, um nach jedem Zug zu prüfen, ob der Spieler einen legalen Zug machen kann (Sollte dem nicht zu sein, hat der Spieler verloren, und der Status wird auf gameOverInit gesetzt, um den Prozess des Game-over-Bildschirms einzuleiten.

Die Funktion funktioniert, in dem sie alle 4 Richtungen mit der Funktion `moveTest()` ausführt, und schaut, ob das Board dasselbe ist wie zuvor -> also die Richtung nicht gültig ist, zum Bewegen.

Sollte nun in allen 4 Richtungen das Board vor, und nach dem Bewegen, dasselbe sein, gibt die Funktion `False` wieder, und ansonsten `True`.

4.12.2023

Heute habe ich ein paar Bilder gezeichnet. Darunter einen Game-Over screen, einen Startbildschirm und einen Siegbildschirm.

5.12.2023

Heute habe ich die Funktion `drawBoard()` geschrieben, die immer das aktuelle Spielfeld im Fenster zeichnet. Dafür habe ich die Original-Farben des Spiels geholt, und in ein Dictionary in meinem config.py gepackt. Dafür brauchte ich auch die Funktion `setup_tiles()`, um alle pygame-Surfaces zu erstellen, sodass ich sie in `drawBoard()` nur noch mit fenster.blit() an die richtige Position (welche sich aus der Position der Zelle, und der Größe pro Zelle (in px) ergibt) einfügen muss. Um die Surfaces zu erstellen, brauchte ich die Funktion `draw_text_in_box()`, um einen größtmöglichen Text in das Rechteck zu schreiben.

7.12.2023

Heute habe ich die Funktion `MaxWert()`, geschrieben, welche das größte tile, dass auf dem Spielfeld liegt, wiedergibt. Diese Funktion benötige ich, um zu prüfen, ob der Spieler schon gewonnen hat (Also ein 2048-tile auf dem Spielfeld liegt.) Die Funktion funktioniert, in dem sie jede Zelle durchgeht, und den größten Wert zurückgibt.

8.12.2023

Heute habe ich meine Gameloop in main.py aufgeräumt. Dazu habe ich z. B. eine Variable (int) direction erstellt, die in der `inputLoop()` geändert wird, falls eine Pfeiltaste gedrückt wird, sodass das Board in der `logicLoop()` in die richtige Richtung bewegt werden kann.

Außerdem habe ich eine Overworld gezeichnet, also die „Karte“, auf der man das jeweilige Level, welches man spielen möchte, anklicken kann.

9.12.2023

Heute habe ich die Klickpositionen für den Levelselect geschrieben. Dies tat ich in dem ich pygame-Rechtecke erstellt habe, und gemeinsam mit einem String, um anzugeben, zu welchem Level das jeweilige Rechteck gehört, in einer Liste gespeichert.

In main.py prüfe ich jetzt beim Drücken der linken Maustaste, ob die Position des Mauszeigers in einem der Rechtecke liegt. Falls also ein Level angeklickt wird, wird das jeweilige Level ausgegeben.

12.12.2023

Heute habe ich die Hitboxen benutzt, und somit das erste Level spielbar gemacht, beim Anklicken.

Wenn also nun ein Level auf der Overworld angeklickt wird, wird der Status auf das Initialisieren des jeweiligen Levels gesetzt (also z.B. „`level1Init`“). In der `logicLoop()` wird nun das Level gestartet, mit der Funktion `initLevel()` und dem zu dem Level zugehörigen Board, welches in einem Dictionary gefunden werden kann.

Außerdem habe ich meine globalen Variablen, in der `mainLoop()` gelöscht und durch `import as cfg` ersetzt, da dies das Programm wesentlich übersichtlicher macht, und die globals wegfallen (weshalb ich nicht immer beim Benutzen einer neuen Variable, diese als global definieren muss).

15.12.2023

Heute habe ich hauptsächlich die `initLevel()` Funktion erweitert, sodass nun nicht nur das Board dem jeweiligen Level angepasst wird, sondern auch die Höhe, die Breite, und somit auch die gesamte Höhe und Breite des Fensters.

16.12.2023

Heute habe ich die `countScore()` Funktion implementiert, die nach jeder Bewegung zum Score hinzuaddiert. Der Score in 2048 gibt an, wieviel gemerged wurde, d.h. wenn man eine 8 in eine 8 schiebt, wird 16 zum Score addiert. Die `countScore()` Funktion führe ich immer nach einer Bewegung und vor einem `merge()` aus, damit sie die markierten(negierten) Zahlen, welche sich im `merge()` verdoppeln, vor dem `merge()` ansehen kann. Dazu musste ich einwenig die `move()` und `up()` Funktion umschreiben. Außerdem habe ich eine `moveTest()` Funktion erstellt, die nur verwendet, um den `move()` zu testen, also z.B., wenn getestet wird, ob es einen möglichen Zug gibt. In `moveTest()` wird nichts zum Score hinzuaddiert (also `countScore()` nicht aufgerufen)

Außerdem habe ich die `drawUiinGame()` Funktion gefüllt, und mir dafür ein Interface ausgedacht. Oben im Interface soll die aktuelle Punktzahl, als auch das Ziel des jeweiligen Levels zusehen sein.

19.12.2023

Heute habe ich pro Level noch mehr Attribute in Dictionaries geschrieben als nur das Board. Darunter die Größe pro tile, die Größe zwischen den tiles, den Text, der in der UI oben angezeigt werden soll, den Gamemode, wobei ich bisher nur den gamemode `maxWertTile` habe, was bedeutet, dass man irgendein tile (z.B.2048) erreichen muss, um zu gewinnen.

Außerdem habe ich einen Gameover, und einen Win-screen gezeichnet.

29.12.2023

Heute habe ich die Dictionaries, für die Levelspezifischen Informationen geändert. Während es zuvor noch viele dictionaries waren (z.b. size, board, etc), die jeweils pro Level einen Schlüssel hatten, habe ich jetzt ein Dictionary pro Level, in dem alle Informationen enthalten sind. Zudem habe ich mir schon 5 Level überlegt, und diese schon eingebaut, indem ich dictionaries erstellt habe, und diese in dem Gemeinsamen Dictionary levels unter dem Schlüssel des jeweiligen Levels (als int) gespeichert. Aufgrund dessen musste ich natürlich Änderungen überall machen, wo ich auf die alten Dictionaries zugreife, insbesondere in `initLevel()`.

Zudem habe ich implementiert, dass die Bilder für Gameover und Sieg angezeigt werden, als auch die Fenstergröße angepasst wird, sodass das Bild reinpasst, wenn man gewinnt oder verliert. Dabei habe ich gemerkt, dass man als Spieler garnicht realisiert warum man gewonnen/verloren hat, wenn der Sieg/Niederlage Screen sofort angezeigt wird, und man das Spielfeld nach dem letzten Zug so gut wie garnicht mehr sieht. Deswegen habe ich versucht einen Timer einzubauen, mit `pygame.time.delay()` und `pygame.time.stop()`, aber aus irgendeinem Grund wird die Unterbrechung unterbrochen, wenn man einen Knopf drückt. Darum möchte ich mich morgen kümmern.

31.12.2023

Heute habe ich mich als erstes um den Timer gekümmert. Dazu habe ich mir eine Timer-Klasse kopiert und verändert, den ich früher schon einmal für ein Projekt geschrieben hatte. Der Timer funktioniert folgendermaßen:

- Variable `timeout`: Variable, die beim initialisieren mitgegeben wurden kann, sagt aus wie lange die Pause sein soll (in ms)
- `Current`: ist die aktuelle Zeit, seit Start des gesamten Programms. Dafür benutze ich die Funktion `pygame.time.get_ticks()`
- `Self.timeout`: der Zeitpunkt (in ms, seit Programmstart), wann der Timer abläuft. Offensichtlich `timeout` addiert mit `current`.

Ab da an ist es relativ selbsterklärend. Mit der Funktion `wait()` gibt der Timer `True` wieder, wenn die Zeit schon abgelaufen ist -> Also der aktuelle Zeitpunkt höher ist als `self.timeout`.

Dies benutze ich nun anstatt den alten Pygame-Timern in meinem main-file. Dafür brauchte ich auch eine Variable `timer_set`, um zu prüfen, ob schon ein Timer gesetzt wurde, damit ich nicht an derselben Stelle jeden Frame einen neuen Timer zu erstellen.

Dannach habe ich mir überlegt, auf der Overworld Schlößer über Level zu malen, die noch nicht freigeschaltet sind. Dafür benutze ich eine Variable `best_level` welche immer den Wert des höchsten geschafften Levels beinhaltet (Am Anfang also 0). Dann male ich unter alle schon geschafften Level (`<=best_level`) ein Gesicht, dass ich gemalt habe, und unter alle, die aktuell noch nicht freigeschaltet sind (`>= best_level +1`) ein Schloss.

Zwischendurch habe ich noch die Farbwerte für die Gegner in mein Dictionary für die Farben eingefügt. Die Gegnertiles sollen, im Gegensatz zu den roten Spielern, einen Grün-blau Ton haben. Diesen habe ich schon beim Zeichnen der Bilder benutzt, deswegen musste ich die meisten rgb-Werte nur noch kopieren.

Außerdem habe ich versucht die einzelnen Zellen meines Boards nicht nur mit Integern zu füllen, sondern mit Listen, damit ich weitere Werte hinzufügen kann, wie z.B. ob es ein Gegner ist, etwas, dass auf dem Boden liegen bleibt, etc. Dafür musste ich natürlich überall, wo ich dieses Board benutze, ändern, wie insbesondere in der `up()` Funktion.

1.1.2024

Heute habe ich zuerst das System mit den Listen in den einzelnen Zellen umgestellt auf Dictionaries. Ich denke dies ist einfach wesentlich einfacher zu ändern, falls ich einen neuen Parameter hinzufügen möchte, oder entfernen möchte, als auch viel übersichtlicher bei der Anwendung. (Damit habe ich auch gestern schon begonnen.)

Die Dictionaries bestehen aktuell nur aus den Schlüsseln `tile_numb` -> für den aktuellen Wert der Zelle, und `fraction` -> Fraktion der Zelle, also Gegner, Spieler und `none` für die 0.

Ich habe mir überlegt, dass ich für alle Zellen, die eine 0 am Anfang haben sollen, ein template Dictionary erstelle (mit dem Wert 0 und ohne Fraktion), welches ich in einer Funktion in jede freie Zelle reinkopiere (mit `copy.deepcopy()`, was ich weiter oben schon beschrieben habe.) Für die anderen Zellen, die ich vorher festlegen möchte, benutze ich das Dictionary unter dem Schlüssel `preset_tiles` in jedem Level-Dictionary. Dort habe ich Dictionaries, mit passenden Parametern für `tile_numb` und `fraction`, unter Koordinaten Paaren (`(x, y)`) als Schlüssel.

Als nächstes habe ich Gegner eingebaut. Dazu habe ich in der `up()` Funktion bei einer Kollision von 2 tiles, die nicht dieselbe Fraktion haben (und wo `fraction != none` ist) folgendes gemacht:

- Wenn der Gegner stärker ist:
  - Dahin wo hinbewegt wird, wird das Dictionary vom Gegner hin kopiert.
- Wenn der Spieler stärker ist:
  - Dahin wo hinbewegt wird, wird das Dictionary vom Gegner hin kopiert.
- Wenn beide gleichstark sind:
  - Dahin wo hinbewegt wir, wird das Dictionary vom Gegner hinbewegt, und die Nummergröße verdoppelt

Bei dem letzten Punkt überlege ich noch das zu ändern, da es möglicherweise nicht intuitiv sein könnte, wenn der Gegner „den Kampf gewinnt“ und sich verdoppelt, auch wenn ich denke, dass dies ein interessantes Element sein könnte.

Für das spawnen des Gegners habe ich ein Element in jedes Level-Dictionary hinzufügt, dass die Prozentchance, dass `createRandom()` als neues tile einen Gegner erstellt.

2.1.2024

Heute wollte ich hauptsächlich neuen Content erstellen, welches ich später beim Designen meiner 40 Level (Wahrscheinlich mache ich doch nur 20, obwohl ich auf meiner Overworld Platz für 40 hätte, da ich denke es ist extrem schwer 40 einzigartige Level zu kreieren.) benutze, und nicht mehr großartig was an dem System, wie das Spiel funktioniert umschreiben, da ich denke, dass dies so weit fertig ist.

Als erstes habe ich Gesichter gemalt, die ich dann in `drawBoard()` auf die Gegner-tiles male, damit sie sich noch ein wenig mehr von den Spieler-tiles unterscheiden.

Als zweites habe ich Wand-tiles geschrieben, welche auf dem Boden liegen, und durch die tiles nicht durchgehen können. Die Funktion habe ich natürlich in `up()` geschrieben, da dies der Hauptbestandteil der Bewegung ist.

Als drittes Feature heute habe ich ein tile geschrieben, dass genau wie die Wand auf dem Boden liegen bleibt, und das tile (egal ob Gegner oder Spieler), welches sich als erstes auf es drauf bewegt, verdoppelt. Auch die Funktion dazu ist größtenteils in `up()` zu finden. Es spawnt zuffällig, genauso wie die Gegner nach der Prozentchance im Level-Dictionary.

4.1.2024

Als erstes habe ich ein Tile zum Halbieren geschrieben, welches am Boden liegt. Es ist genauso, wie das zum Verdoppeln von Vorgestern, nur zum Halbieren, deswegen kann ich dazu jetzt nicht so viel schreiben.

Als nächstes habe ich ein Feld angefangen zu implementieren, welches IMMER auf dem Boden liegen bleibt, um es als Sieg-feld zu benutzen (Also das man nicht gewinnt, wenn man 2048 erreicht hat, sondern nur wenn man 2048 erreicht hat und dies auf dem Sieg-feld liegt.). Die größte Schwierigkeit damit war, dass es immer auf dem Boden liegen bleiben soll, und somit auch, nachdem ein tile über dieses Feld gelaufen ist. Das habe ich gelöst, indem ich das Dictionary mit den Koordinaten als Schlüssel für das Sieg-feld nicht in `preset_tiles`, sondern unter dem neuen Schlüssel `stay_on_floor_tiles speicher`. Was das Zeichnen und zum Board Hinzufügen angeht, auch wenn ein tile drüber gelaufen ist, habe ich dieses Problem gelöst, indem ich in der `move()` Funktion, nachdem alles bewegt wurde, jedes tile, in `stay_on_floor` an dessen Position hineinkopiere, sollte diese nicht schon bewegt worden sein.

Um jetzt unter dem neuen gamemode zu prüfen, ob der Spieler gewonnen hat, habe ich die Funktion `check_sieg_feld()` geschrieben, die prüft, ob ein tile, dass gleich, oder größer als das Ziel ist auf dem Sieg-feld liegt. Sollte das so sein wird natürlich wie gewohnt der Status auf siegInit gesetzt, sodass die Fenstergröße angepasst werden kann, und das Sieg-Bild eingefügt werden kann.

5.1.2024

Heute wollte ich das Spiel, bis auf ein paar Verfeinerungen, fertigstellen. Dazu musste ich eigentlich nur noch die 20 Level erstellen, da ich schon alle notwendigen Features habe. Dazu habe ich mir eine Liste aller Level erstellt, damit ich den Fortschritt, den der Spieler durchfühlt, sehen kann, bevor ich anfange es zu implementieren:

Level 1:
3x3, 64

Level 2:
3x3, 128

Level 3:
4x4, 256

Level 4:
4x4, 512

Level 5:
4x4, 512, wand in ecke

Level 6:
4x4, 512, 2 wände am Rand/in der Mitte (0,1) (1,1)

Level7:
5x4, 512, Gegner

Level8:
6x4, 1024, mehr Gegner

Level9:
6x4, 1024, Gegner, und ein paar Wände als Linie ins Level

Level10:
Schwierige zwischenstage-nur ein schlangenförmiger weg. 32 ist das Ziel. Keine Gegner

Level11:
4x4, 512. Verdopplungen eingeführt.

Level 12:
8x4, 1024. viele Verdopplungen und Wände. Keine Gegner

Level13:
6x4, 512, Verdopplungen werden viel genutzt und weitergedacht. Mit Gegnern

Level14:
6x4, 128, halbieren wird eingeführt

Level15:
8x4, 1024, Alles außer Gegner. Fokus auf halbieren.

Level16:
Halbieren mit vielen Gegnern

Level17:
Schweres Level mit allen Features bis jetzt.

Level 18:
Einführung Sieg-feld

Level 19:
8x4, sieg-Feld mit Gegnern

Level20:
10x4 Sehr schwer, Sieg-feld hinter Wänden

Nun habe ich alle Level erstellt, indem ich für jedes Level ein Dictionary erstellt habe, mit dem Spielfeld, dem Spielmodus, dem Ziel, vorbestimmte tiles, wie bspw. Wände, etc.

Nach dem Polieren der Level, sowie kleinen Änderungen (darunter die Farben der Gegner), brauchte ich eine Funktion für das. letzte Level. Dort wollte ich nämlich nicht, dass neue tiles auf der unteren Reihe spawnen, es sei denn sie haben keinen anderen Platz mehr, da das Level ansonsten schlichtweg unmöglich wäre.

Dafür habe ich im Dictionary des letzten Levels eine Liste, mit Koordinaten, unter dem Schlüssel „`blocked`" gespeichert. `CreateRandom()` habe ich nun so umgestellt, dass dort keine neuen tiles spawnen, wenn es nicht anders geht.

Außerdem habe ich heute neue Levelselect Bilder gemalt, da ich die alte zweite Seite nicht mehr brauche, und die erste angepasst habe. Dazu zählt, dass ich einen „unlock all" Button in der linken oberen Ecke eingefügt habe, der alle Level von Anfang an spielbar macht.

6.1.2024

Heute habe ich eine kleine Idee eingebaut, welche ich in den letzte paar Tagen hatte. Undzwar habe ich eine zweite Seite des Levelselects, wo man das orginale Spiel (Also unendlich lang, ohne Abbruchbedingung, und mit `max_moves_per_tile =-1`, sodass die tiles sich immer bis zum Rand bewegen) in verschiedenen Größen des Spielfelds spielen kann. Zudem wollte ich dies eigentlich noch machen mit Gegnern, aber aus irgendeinem Grund Funktionieren die Gegner nicht richtig, wenn sie sich bis ganz zum Rand bewegen, und nicht nur um immer ein Feld. Das zu beheben habe ich jetzt keine Zeit mehr.

Falls sie nicht alle Level spielen möchten, habe ich hier eine Liste mit Leveln, die ein neues Feature einführen:

- Level 5 (Mauer)
- Level 7 (Gegner)
- Level 11(Verdopplungen)
- Level 14(Halbieren)
- Level 18(Sieg-feld)

Und hier alle Level, die meiner Meinung nach diese Elemente am besten weiterdenken, und es zu einem, wie mein Ziel am Anfang

- Level 10
- Level 13
- Level 17
- Level 20

Zudem habe ich noch weiter Ideen, diese Spielideen zu expandieren, wie das Implementieren von Böden, die tile in eine andere Richtung weitergeben, Portale an den Seiten, wie in Pacman, oder Bomben, welche auf dem Boden liegen, und bei Kontakt alle tile in einem bestimmten Umkreis auslöschen. All diese Sachen habe ich zeitlich nicht mehr geschafft, wären aber ohne große Änderungen am Spiel implementierbar, da ich die Gameengine von Anfang an sehr offen geschrieben habe.