Frankfurt University of Applied Sciences

International Business Information Systems

Betriebssysteme und Rechnernetze, 2. Semester

Prof. Dr. Christian Baun

Werkstück A7

Python-Implementierung eines Maze-Spiels

(Maze-Generator, Pathfinder)

Vorgelegt von:

Stella Malkogiannidou, 1091458

Email: malkogia@stud.fra-uas.de

Samira Maryan Hersi, 1264751

Email: hersi@stud.fra-uas.de

Amaal Mohamed, 1265365

Email: amaalmoh@stud.fra-uas.de

I. Inhaltsverzeichnis

II. Listingverzeichnis

III. Abbildungsverzeichnis

IV. Tabellenverzeichnis

1. Abstract

Die vorliegende dokumentierte Gruppenarbeit zielt darauf ab, einen ausführlichen Einblick über die Vorgehensweisen, Herausforderungen sowie Problemlösungen, während der Projektentwicklung zu verschaffen. Das fertig implementierte Programm ist unterteilt in der Konsole- und der graphische-Ausgabe, welche sich voneinander kaum unterscheiden. Hinsichtlich der graphischen Implementierung entschieden wir uns für das Python-Modul „PyGame[1]“. Das Programm startet in der Python-Shell (Konsole) und nach Eingabe valider Achsenwerte erfolgt die Ausgabe des zufälligen “perfect Maze” in der Konsole- sowie der graphischen Ausgabe.

2. Einleitung

Die ersten Maze-Spiele erschienen etwa in 1973. Das erste 3D-Maze-Spiel für Homecomputer Sinclair ZX81 wurde im Jahr 1981 von Malcolm Evans entwickelt.[2] Das Maze ist ein Rätsel, bei dem sich der Spieler über Tastatureingaben interaktiv bewegt, bis er das Ziel erreicht. Während der Interaktion navigiert der Spieler durch das Labyrinth und markiert währenddessen die einzelnen besuchten Felder. Dabei ist es wichtig, dass der Spieler nur freie Felder betritt. Darüber hinaus hat dieser noch die Möglichkeit, das Maze-Spiel vorzeitig zu beenden. Bei der Generierung des Labyrinths verhindert das Programm, dass das zufällig ausgewählte Start- und Zielfeld an der gleichen Position liegen. Die im Programm angewendeten Konstanten sind durch einen Import in den jeweiligen Modulen global definierte Variablen. Dies erleichterte das Ändern der Konstanten und diente zu einem verständnisbaren Code. Besonders bei langen Textausgaben erlaubte die Auslagerung kürzere Methoden zu implementieren.

3. Aufbau und Entwicklung des Programms

Zu Beginn des Projekts führte das Team eine Anforderungsanalyse durch, um die Aufgabenstellung in logische Teilbereiche zu unterteilen. Das Programm besteht aus den folgenden Modulen *model.py*, *mazespiel.py*, *algo.py* und *konstanten.py*. Im *model.py* sind alle Datenstruktur-Klassen definiert. Zum algo-Modul gehören die Klasse MazeGenerator und die Klasse PathFinder (Lösungspfad). Im mazespiel.py Modul befindet sich das Hauptprogramm in der *Konsole*-Klasse, das Spiel in der MazeSpiel-Klasse und die *main()* als Modul-Funktion zum Starten des Programms. Das Team entschied sich für den rekursiven-depth-first-Backtracking-Algorithmus (rDFB-algo)[3]. Bevor der rDFB-Algo startet, ist jeder Knotenpunkt mit dessen Nachbarn durch eine Kante verbunden, sodass ein gitterförmiges

[1] Siehe Pygame Library nutzen zum Spiele programmieren in Python. (Zuletzt geprüft am: 10.06.2021).
[2] Siehe 3D Monster Maze (Zuletzt geprüft am: 20.06.2021).
[3] Siehe Maze generation algorithm, Rekursive Implementation (Zuletzt geprüft am: 24.06.2021).

Kantennetzwerk vorliegt. Zu Beginn des Projekts betrieb das Team ein Reverse Engineering, um die Animation des rDFB-Algo im Hinblick der Eigenschaften und des Verhaltens zu untersuchen[4]. Unter Zuhilfenahme von Microsoft Excel betrachteten die Mitglieder die Excelzellen als die Kanten und die Kantenüberschneidungen als die Knotenpunkte (Koordinate). Beim Zeichnen des Anfangszustand des Labyrinths fiel auf, dass das Modell lediglich 2 Kanten pro Knotenpunkt benötigt, um fast das komplette Kantennetzwerk zu erstellen.

Die Abbildung 1 zeigt die gesammelten Eigenschaften der Reverse-Engineering-Analyse an. Dabei stellen die Kantenfarben Gelb und Pink den Anfangszustand des Labyrinths dar. Die unterschiedliche Einfärbung zeigt zudem noch an, zu welcher Koordinaten-Instanz (k-Instanz) die entsprechende horizontale Nord- und die vertikale Westkante gehört. Aus den eingegebenen Achsenwerten resultierte daraus ein Labyrinth, dessen rechtes und unteres Rand offen erschien. Dies wird in der Abbildung 1 mit der rot gestrichelten Linie illustriert. Aus diesem Grund initialisiert der Algorithmus in der *Maze*-Klasse ein 2D-Array mit (yaxis+1, xaxis+1).

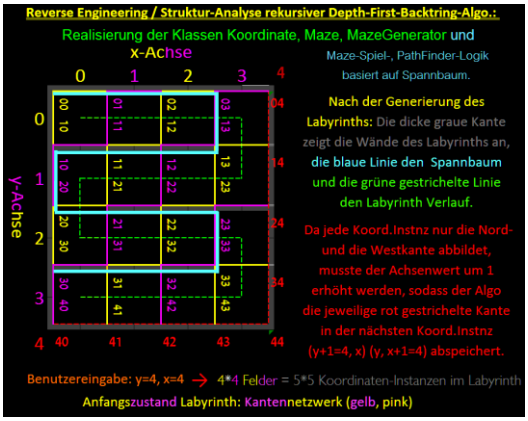


Abbildung 1: Reverse Engineering und Strukturanalyse

Um aus dem Anfangszustand des Labyrinths ein zufälliges Gebilde erzeugen zu lassen, kommt der rDFB-Algo zum Einsatz. Zu Beginn der Generierung wählt der Algorithmus zufällig den nächsten Nachbarn aus der neighbours-Liste der aktuellen k-Instanz aus und löscht diese Auswahl aus der Liste, um ein erneutes zufälliges Auswählen dieses Nachbarn beim Backtracking zu verhindern. Neben den Indices für das nächste Feld, erhält der Algorithmus auch den Schlüssel “h” oder “v”, um die richtige Kante aus dem Attribut *kanten* der entsprechenden k-Instanz zu löschen.

Die Abbildung 2 auf Seite 3 veranschaulicht die Kantenzugehörigkeit durch die grellgrünen oder dunkelroten Farben. Befindet sich der zufällig ausgewählte Nachbar links oder über das aktuelle Feld (dunkelgrün), so gibt die grellgrüne Kante an, dass der übergebene Kanten-Wandtyp-Schlüssel aus dem Attribut kanten der aktuellen k-Instanz zu löschen sei. Befindet sich jedoch der Nachbar in der

[4] Siehe Maze Generation, Algorithm Recap (Zuletzt geprüft am: 24.06.2021).

entgegengesetzten Richtung unten oder rechts vom aktuellen Feld, so löscht der Algorithmus den übergebenen Kantenschlüssels “h” oder “v” aus dem Attribut *kanten* der k-Instanz des Nachbarn.

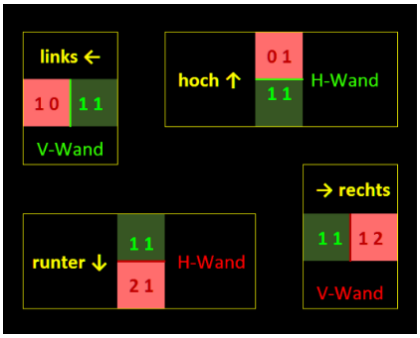


Abbildung 2: Kantenzugehörigkeit zur Löschung der entsprechenden Wände

Um die Implementierung des rDFB-Algorithmus prüfen zu können, erfolgte zunächst die Ausgabe des Labyrinths in der Konsole, welche die Abbildung 2 veranschaulicht. Das linke Bild der Abbildung 2 zeigt eine ältere Version, welche als Proof-of-Concept diente. Nachdem alle Fehler korrigiert waren, überlegte das Team sich ein Verfahren, um aus 16 verschiedenen Zeichen das richtige für jede Spalte und horizontale Zeile auszuwählen, sodass eine “perfekte” Ausgabe eines “perfect” Maze in der Konsole erscheint. Auf das Verfahren geht das Team bei der Klasse Maze detaillierter ein.

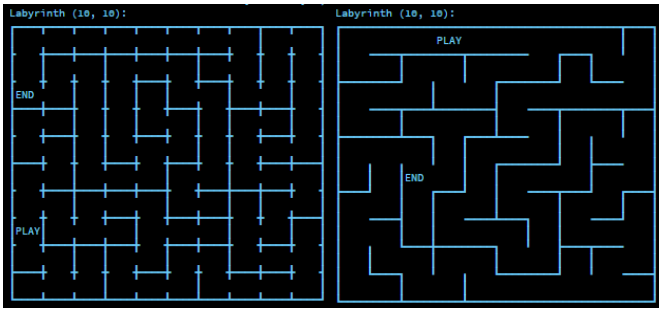


Abbildung 3: Ursprüngliche & verbesserte Konsolenausgabe des Labyrinths

Durchgeführte Tests ergaben, dass bei Achsenwert-Angaben jenseits von x=33, y=30 dazu führten, dass das Programm aufgrund eines RecursionError (Pufferüberlauf-Fehler, Stackoverflow) abstürzt[5]. Die Chance eines RecursionErrors steigt, sobald die Feldanzahl im Labyrinth den Wert 999 übersteigt. Solche Fehler lassen sich nur durch eine iterative Implementierung lösen[6]. Die Anzahl der Felder im Labyrinth erhöhte sich auf 3600 Felder, indem das Programm bis zu 500-mal versucht ein Labyrinth zu erzeugen. Durch try-except fing das Programm die RecursionError-Fehler ab und unterrichtete den Benutzer über die Anzahl an Versuche. Zur Implementierung des Maze-Spiels entschied sich das Team für das Pygame Framework. Im Gegensatz zu textbasierten (ncources) und graphischen Alternativen (TKinter, Pygame,

[5] Siehe Python - Handling recursion limit (Zuletzt geprüft am: 20.06.2021).
[6] Siehe Maze generation algorithm, Iterative Implementation (Zuletzt geprüft am: 23.06.2021).

Turtle) gibt es bei Pygame genügend Literatur und ist speziell für Spiele konzipiert. Nach der Implementierung der Benutzereingabe zu den Achsenwerten und dem Erlernen des Pygame-Frameworks, kam die Implementierung der Spielersteuerung des Maze-Spiels durch die Angaben der Pfeiltasten und bei den Gamern etablierten Tastenfolgen “w, a, s, d”. Als nächstes integrierte das Team die Logik der Spielsteuerungstasten, die lediglich prüft, ob der Spieler in das gewünschte Feld sich hinbewegen darf.

Darüber hinaus erfolgten Erweiterungen, wie der *PathFinder* oder die Animation des Spannbaums. Der parametrisierte Start als Teil der Aufgabenstellung erfolgte ganz zum Schluss. Die Herausforderung bestand darin, auf die Idee zu kommen, dass jeder rekursive Aufruf an sich eine Schleife darstellt und dass die Rekursion auf einem Stack basiert. Erschwerend kam hinzu, die Absicherung und Wiederherstellung der Variablen-Zustände des vorigen Funktionsdurchlauf als iterative Variante zu implementieren. Letztendlich verhalf der PseudoCode* zur Erkenntnis, dass der Zugriff auf die Information, welche Nachbarn der Koordinateninstanz bereits besucht waren, durch die Definition der neighbours-Liste innerhalb der Koordinaten-Instanz selbst, möglich ist. Anschließend erfolgte die Umsetzung der Klasse *PathFinder* als nicht Teil der Aufgabenstellung, Die erleichterte Umsetzung begründet sich durch den iterativen DFB-Algorithmus, welcher auch bei der Generierung des Labyrinths zum Einsatz kam (*createMaze*). Die Datengrundlage des PathFinder-Algorithmus (*findPath*) ist der *spanning3*, welcher auch für die Validierung der Spielerbewegung im Maze-Spiel zur Verwendung kommt. Das Suchen nach der Zielfeld-Koordinate verlauft ebenso durch zufällige Auswahl und die Lösung der zufälligen Auswahl, kam auch im *findPath* zum Einsatz. Allerdings erfolgt eine 1:1 Kopie, bevor die Berechnung des Lösungsweges startet. Dadurch bleibt der *spanning3* intakt, wodurch das Maze-Spiel, nach der Berechnung des Lösungsweges, weiterhin spielbar bleibt.

4. Klassen

Im weiteren Verlauf der Dokumentation sind alle implementierten Programmklassen in logischer Reihenfolge beschrieben.

4.1 Koordinate-Klasse

Die x- und y-Werte repräsentieren die Indices der Stelle im 2D-Array *labyrinth* der Klasse *Maze.* Die *laenge* gibt die einzelnen Kantenwandlänge an. Die vorigen Attribute dienen zur Berechnung des Pygame-Rect-Objekts[7] für das *marker*-Feld mit der entsprechenden Farbe aus markercolor sowie für *solutionMarker*-Feld mit der entsprechenden *solutionMarkerColor* zur graphischen Ausgabe. Das Programm initialisiert zunächst ein leeres *kanten*-Dictionary zur späteren Abspeicherung der Schlüssel “h” und “v” und als Wert ein berechnetes Pygame.Rect-Objekt für die graphische Ausgabe. Die neighbours-Liste beinhaltet die Indices der direkten Nachbarn dieser K-Instanz mit dem entsprechenden

[7] Siehe Pygame Tutorial, Work with rectangles ((Zuletzt geprüft am: 20.06.2021).)

kanten-Schlüssel “h” oder “v”. Der isVisited-Boolean gibt an, ob diese K-Instanz bereits besucht wurde oder nicht.

4.2 Maze-Klasse

Die Klasse Maze beinhaltet das Labyrinth als ein 2D-Array. Bei der Initialisierung des 2D-Arrays mit der Größe yAchse+1, xAchse+1 instanziiert das Programm für jede Stelle eine Objektinstanz der Klasse Koordinate. Der Algorithmus berechnet einen langen Ausgabestring zur Repräsentation des Labyrinths durch die Unicodes. Wie die Abbildung 4 veranschaulicht, durchläuft das Programm jede Zeile des Labyrinths zweimal, um in der einen Zeile die horizontalen und in der nächsten Zeile die vertikalen Kanten abzubilden. Der Algorithmus gibt für jede horizontale Kante 5 Zeichen aus, wovon das erste Zeichen dieser Kante dynamisch errechnet wird. Während das Programm in der horizontalen Kante nach dem Schlüssel “h” sucht, prüft das Programm in der vertikalen Kante nach der Existenz des “v” Schlüssels. Wenn ein “h” oder “v” Schlüssel existiert, gibt das Programm die entsprechende Kante an. Im Gegensatz dazu, gibt das Programm in der vertikalen Kante den *marker*, wie Start- und Zielfeld an.

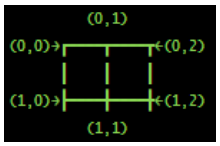


Abbildung 4: Zeichnung der horizontalen & vertikale Kanten

Die *getZeichenCode*-Methode bildet einen Binärkodierungsstring mit der nachfolgenden Definition:

- **Nach Rechts:** prüft im *kanten* Dictionary der aktuellen k-Instanz, ob eine Kante nach rechts durch den Schlüssel “h” existiert, wenn ja dann gebe eine “1” zurück, sonst “0”.
- **Nach Unten:** prüft im *kanten* der aktuellen k-Instanz, ob eine Kante nach unten durch den Schlüssel “v” existiert, wenn ja dann gebe eine “1” zurück, sonst “0”.
- **Von Links:** prüft in *kanten* der k-Instanz links von der aktuellen k-Instanz, ob eine Kante von links durch den Schlüssel “h” existiert, sofern diese k-Instanz valide ist. Für Wenn ja, dann gebe eine “1” zurück, sonst “0”.
- **Von Oben:** prüft in *kanten* der k-Instanz über der aktuellen k-Instanz, ob eine Kante von oben durch den Schlüssel “v” existiert, sofern diese k-Instanz valide ist.

Der gebildete Binärkodierungsstring dient als Schlüssel, um das entsprechende Unicode-Startzeichen der horizontalen Kante aus dem Dictionary *wallChar* darzustellen. Wie der Tabelle 1 zu entnehmen ist, entspricht zum Beispiel die vertikale Binärkodierung '1011' das Zeichen '┴'.

Prüfungs-reihenfolge	┼	Kanten-anzahl: 3 ┼ ┴ ├ ┬	Kanten-anzahl: 2 ┘ └ ┐ ┌ | -	Kanten-anzahl: 1 ╵ - ╷ -
nachRechts: -	1 0	0 1 1 1	0 1 1 0 0 1	0 1 0 0
nachUnten : |	1 0	1 0 1 1	0 0 1 1 1 0	0 0 1 0
vonLinks *: -	1 0	1 1 0 1	1 0 0 1 0 1	0 0 0 1
vonOben * : |	1 0	1 1 1 0	1 1 0 0 1 0	1 0 0 0

Tabelle 1: Binärkodierung der Startzeichen für die horizontale Kanten des Labyrinths

Die Funktion *isValid_and_isNotVisited* prüft, ob die übergebene Koordinate *y, x* innerhalb des Labyrinths sich befindet und ob die k-Instanz an der Stelle *y, x* im 2D-Array Labyrinth noch nicht besucht wurde.

4.3 MazeGenerator-Klasse

Diese Klasse erzeugt ein zufällig generiertes "perfect"-Maze (Maze). Die Erstellung eines zufällig generierten Labyrinths setzt die Objektinstanziierung der Klasse *Maze* voraus, worin Zugriff über die Objekt-Instanz *maze* auf das 2D-Array *labyrinth* besteht. Bevor der Vorgang mit der Generierung des *Maze* beginnt, bildet die *createWalls* Methode den Anfangszustand des Labyrinths. Der Schlüssel wird durch ein Tupel[*1] mit dem y- und x-Wert der entsprechenden *Koordinate*-Instanz angegeben Der Wert des Schlüssels ist eine Liste*[2], bestehend aus einer weiteren Liste[*3] mit den Indices, welche die *Koordinaten.Instanz* im 2D-Array *labyrinth* an der entsprechenden Indices-Stelle angibt:

Dict[Tuple[int,int]: List[List[int,int]]]

***1 *2 *3**

Zum Beispiel: **{(6, 8): [[6, 9]], (6, 9): [[6, 8], [7, 9]], (7, 9): [[6, 9], [8, 9]],…}**

In der Abbildung x erstellt der Algorithmus in der *createWalls* ein Kantennetzwerk. Dabei legt das Programm für jede K-Instanz im *kanten* Dictionary den Schlüssel "h" und "v" an und als Wert die berechnete Kantendimension Pygame.Rect-Objekt für die graphische Ausgabe der Labyrinth-Wände.

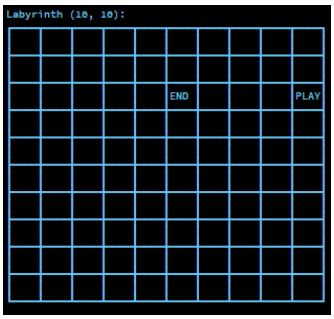


Abbildung 5: Erstellung des Kantennetzwerkes als Anfangszustand des Labyrinths

Zur Implementierung des iterativen Algorithmus orientierte sich das Team am folgenden Pseudocode[8]:

1. Markiere die übergebene Zelle *k* als besucht und lege (*push*) diese auf den Stapel (*stack).*
2. Solange der *stack* nicht leer ist:

2.1 Entnehme die zuletzt gepushte Zelle aus dem *stack* (*popp*) und mache diese zur aktuellen *cell.*

2.2 Wenn die aktuelle *cell* über Nachbarn verfügt (*Koordinate.neighbours*-Liste), die noch nicht besucht wurden:

2.2.1 Dann lege die aktuelle Zelle *cell* auf den Stapel *stack.*

2.2.2 Wähle eine zufällige Nachbarszelle aus der *neighbours*-Liste der aktuellen *cell* aus, die noch nicht besucht wurde und mache diese zur *nextCell.*

2.2.3 Entferne die Wand, die zwischen der *cell* und der zufällig ausgewählten *nextCell* liegt.

2.2.4 Markiere die ausgewählte *nextCell* als besucht und lege diese auf den Stapel *stack.*

Während der Generierung des Labyrinths (#2.2.3 - #2.2.4) bildet der Algorithmus auch den Spannbaum *spanning3.* Wenn die *nextCell* valide ist und noch nicht besucht wurde, dann legt der Algorithmus den Schlüssel c*ell* an, erweitert die Werteliste mit dem nextC*ell* und anschließend den umgekehrten Weg mit *nextCell* als Schlüssel und fügt dessen Werteliste *cell* hinzu. Der erste Schlüssel ist der Startpunkt der Generierung des Labyrinths.

4.4 Konsole-Klasse

Das Hauptprogramm befindet sich in der *run()* Funktion und läuft solange *running* True ist. Zunächst erfolgt nach einer Menüausgabe die Benutzereingabe der Achsenwerte in der *setXYachsen*. Mittels try-except verhindert das Programm mögliche Falscheingaben, wie statt Ziffern andere Zeichen oder zu viele oder zu wenige Achsenwerte. Die Methode läuft solange, bis der Benutzer valide Achsenwerte eingibt. In einer Validierung der eingegebenen Achsenwerte prüft das Programm, ob die y- und x-Achsenwerte größer 0 sind und mindestens 10 betragen. Durch die Berechnung der lokalen Bildschirmauflösung geteilt durch die kleinste Feldkantenlänge von 4 Pixel ermittelt das Programm für eine vollständige Anzeige des Labyrinths die maximale Anzahl der jeweiligen Achsenwerte. Falls eines der Achsenwerte höher liegen sollte, wie der maximale Achsenwert in Abhängigkeit zur Auflösung, gibt das Programm die maximalen y- und x-Achsenwerte zusammen mit dessen Auflösung als Information aus. Des Weiteren, erfolgt die Berechnung der Feld-Kantenlänge durch die lokale Bildschirmauflösung geteilt durch die Anzahl der jeweiligen Achsenwerte. Unter den 2 Ergebniswerten gilt der kleinere Wert als die Feld-Kantenlänge für die graphische Ausgabe. Darüber hinaus berechnet das Programm den Bildschirmmodus für die graphische Ausgabe des Spiels Die Entscheidung, ob im Fenstermodus oder Vollbildmodus die Ausgabe des Labyrinths erfolgt. hängt davon ab, wieviel Anzeigefläche das Labyrinth auf dem Bildschirm einnimmt.

Zur Generierung eines Labyrinths in der benutzerdefinierten Feldgröße, erstellt das Programm eine neue Objektinstanz der Klasse *MazeGenerator sowie* der Klasse *Player* zur zufälligen Generierung des Start-

[8] Siehe Maze generation algorithm, Iterative Implementation-Pseudocode (Zuletzt geprüft am: 24.06.2021).

und Zielfelds *marker* durch die Übergabe des Strings „PLAY“ und „END “. Das Programm gibt die gemessene Dauer zur Generierung des Labyrinths, sowie dessen Ausgabe in der Konsole aus. Nach der Konsole-Ausgabe des Labyrinths startet das eigentliche Spiel mit einer graphischen Benutzeroberfläche (GUI). Das Programm erstellt dabei eine neue Objektinstanz der Klasse *MazeSpiel*. Nach dem Spiel-Ende erfolgt die Spielerauswertung. Zu guter Letzt gibt das Programm das generierte Labyrinth mit dem Lösungspfad in der Konsole aus.

5. Graphische-Ausgabe (GUI)

Die graphische Ausgabe beinhaltet die Klassen Player, Stack und PathFinder, welche im weiteren Verlauf der Dokumentation beschrieben werden.

5.1 Player-Klasse

In der while-Schleife wird ein neu zufälliges Start- und Zielfeld ausgewählt, solange sich beide Felder an der gleichen Position liegen. Die Spielerbewegung zum nächsten Feld erfolgt nur dann, wenn die Prüfung durch die *isDirectionValid* Funktion positiv verlief. Wenn die Nach-Koordinate in der Werteliste des aktuellen Von-Koordinaten-Schlüssels im Dictionary *spanningTree* existiert, gibt das Programm ein True, sonst False zurück.

5.2 Stack-Klasse

Der Stack (Stapel) arbeitet nach dem Last-In-First-Out-Prinzip (LIFO) und wendet eine Liste an, um den Zugriff auf die Objektinstanzen der Klasse Koordinate zu ermöglichen. Im Gegensatz zu Listen oder Arrays ermöglichen Stapel normalerweise keinen wahlfreien Zugriff auf die darin enthaltenen Objekte. Die Einfüge- und Löschvorgänge werden oft auch als push und pop bezeichnet.[9]

5.3 PathFinder-Klasse

Der PathFinder arbeitet nach dem iterativen rDFB-Verfahren und verwendet als Datengrundlage den im MazeGenerator-Klasse generierten Spannbaum *spanning3.* Allerdings erfolgt eine 1:1 Kopie des *spanning3,* wenn *isDoPathFinder* True ist. In der *findPath* Funktion berechnet der Algorithmus den Lösungsweg wie folgt:

1. Lege die aktuelle Position des Spielers auf den Stapel (*stack).*Solange die *nextCell* Koordinate ungleich die Zielkoordinate ist:
2. Entnehme die zuletzt gepushte Zelle aus dem *stack* (*popp*) und mache diese zur *cur*rentC*ell.*
3. Wenn die Werteliste des Schlüssels *currentCell* nicht leer ist:
4. Lege *currentCell auf den stack.*
5. Wähle eine zufällige *nextCell* aus und entferne diese aus der Werteliste des Schlüssels *currentCell,* um beim Backtracking die zufällige Auswahl der gleichen *nextCell* zu verhindern.
6. Lösche *currentCell* aus der Werteliste des Schlüssels *nextCell*.
7. Lege die *nextCe*ll auf den stack.

[9] Siehe Grundlegendes zur Stapelimplementierung in Python (Zuletzt geprüft am: 22.06.2021).

Nach der Fertigstellung des Lösungspfads erstellt die *solutionPath2Labyrinth* Methode für jede k-instanz im Lösungspfad das Rect Objekt, um den Lösungspfad in der graphischen Ausgabe anzuzeigen. Die Abbildung 6 veranschaulicht den Lösungsweg von der aktuellen Spielerposition bis zum Zielfeld.

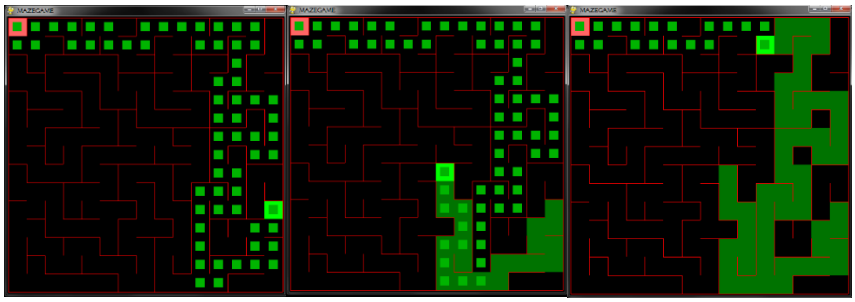


Abbildung 6: Anzeigen des Lösungsweg von der aktuellen Spielerposition

5.4 MazeSpiel-Klasse

In dieser Klasse erfolgt die Implementation des Maze-Spiels. Die Objektinstanz der Klasse MazeSpiel beinhaltet die Achsenwerte, *MazeGenerator und Player*-Instanz und den Anzeigemodus *screentype.*

Die Klasse beinhaltet folgende wichtige Punkte:

- Importieren und initialisieren der Pygame-Bibliothek
- Berechnung der Größe des pygame-Fensters und Begrenzung der Frames pro Second (FPS)
- Menüvorbereitung durch *initMenu*: Menütextzeilen als Bild gerendert.
- Bildung des Start- und Zielfelds durch *markStart_Ziel* für die graphische Ausgabe.

Das folgende Listing gibt den Ablauf des Hauptprogramms *run()* wieder:

- Anzeigefläche durch Übermalen mit *BGCOLOR* zurücksetzen sowie die Benutzerinteraktion.
- *do_drawLabyrinth:* Zeichnen des Labyrinths. Der Algorithmus iteriert durch das 2D-Array und prüft für jede K-Instanz, ob im marker-, solutionMarker-Feld das Pygame.Rect existiert. Bei den Kanten prüft dieser lediglich auf die Existenz des Schlüssels “h” oder “v”, um die entsprechende Kante zu zeichnen.
- In der do_drawMenu Funktion zeichnet das Programm das Menü.
- Wenn *isShowSpaning3* True ist, erfolgt die Animation des Spannbaums.
- Nach dem Zeichen des Labyrinths erfolgt die Aktualisierung sowie das Anzeigen der gesamten Anzeigefläche.
- Das Programm vergleicht die aktuelle Spielerposition mit der Zielfeldkoordinate und falls diese identisch sind, setzt dann Programm *isSuccess* auf True und es erfolgt eine Berechnung der Spieldauer. Dabei wird *isRunning* auf False gesetzt und die while-Schleife beendet.
- In der do_printGameMetrics Funktion erfolgt die Spielauswertung (Gesamtschritt-, valide, invalide Schrittanzahl, Spieler-Effizienz usw.). Zu guter Letzt erfolgt das Beenden des Pygames.

Während des Spiels kann der Spieler die folgenden PC-Tasten betätigen:

- **[q]-, [Esc] - Taste:** Vorzeitiges Beenden des Maze-Spiels.
- **[←, ↑, →, ↓]-Pfeiltasten** oder **[a, w, d, s]-Tasten** zur Spielsteuerung: Das Programm errechnet mit Hilfe der *mody,modx* im Dictionary *directions* die nächste Spielerposition. Bei einer validen Spielerbewegung übergibt das Programm die neu berechnete Position an die *player.setPos(),* sodass diese in der nächsten Runde die aktuelle Position ist. Andernfalls ändert sich die aktuelle Position bei einer invaliden Spielerbewegung nicht. Bei einer validen Bewegung zeigt das Programm die neue aktuelle Position durch die VALIDMOVECOLOR, die alte Position durch PLAYERPATHCOLOR und bei einer invaliden Bewegung die aktuelle Position in der Farbe INVALIDMOVECOLOR an.
- **[F1]-Taste:** Der Algorithmus berechnet, ausgehend von der aktuellen Position des Spielers, den Lösungsweg neu. Vor der Errechnung des neuen löscht das Programm in der *pathFinder.resetMarker* Methode den alten Lösungsweg.
- **[g]-Taste:** bewirkt durch die Neuinstanziierung der Klasse Player eine Neuvergabe des Start- und Zielfelds. Vor der Neuvergabe setzt das Programm sowohl den *Koordinate.marker* als auch *Koordinate.solutionMarker* zurück. In der *markStartZiel()* Funktion erzeugt das Programm für das neu generierte Start- und Zielfeld das Pygame.Rect Feld mit der m*arkerColor*.
- **[k]-Taste:** Die zweifarbige Darstellung der Wände beweist, dass jede Koordinateninstanz im Labyrinth genau zwei Kanten beinhaltet.
- **[m]-Taste:** Bewirkt das Anzeigen bzw. Ausblenden des Spielmenüs**.**
- **[Eingabe]-Taste:** Der Spieler startet die Animation des Spannbaums. Der Algorithmus erstellte im *MazeSpiel.init* bereits eine 1:1 Kopie *sp3copy,* damit das Programm für die Animation den *sp3copy* zerstört. Der Start-Schlüssel-Von-Koordinate für die Animation ist die Startkoordinate der Labyrinth-Generierung. Aus der Werteliste des Von-Koordinate-Schlüssels entfernt das Programm die zuletzt hinzugefügte Nach-Koordinate, sodass diese im nächsten Durchlauf der neue Von-Koordinate-Schlüssel ist. Solange der Algorithmus "vorwärts" läuft, erzeugt er für jedes m*arker*-Feld das Pygame.Rect mit der GENERATOR_*COLOR.* Felder, worin bereits ein *Pygame.Rect* existiert, übermalt der Algorithmus mit der *BACKTRACKER_COLOR.* Der Animationseffekt erfolgt durch die Begrenzung der angezeigten Bilder pro Sekunde. Die Animation endet, sobald die Werteliste des nächsten Von-Koordinate-Schlüssels leer ist. Wie aus der Abbildung 7 ersichtlich ist, zeigt die Animation des Spannbaums zum einen die Labyrinth-Generierung und beweist gleichzeitig, dass alle Labyrinth-Felder begehbar sind.

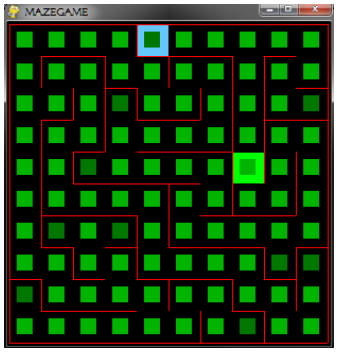


Abbildung 7: Animation des Spannbaums in der graphischen Ausgabe

6. Fazit

Insgesamt setzte sich das Team mit der Implementierung des Projekts 6 Wochen intensiv aus. Für die erfolgreiche Umsetzung der Projektarbeit betrieb jedes Teammitglied sorgfältige Literatur-Recherchen, um reichliche Ideen zu sammeln. Da eines der Gruppenmitglieder, während seiner Werkstudententätigkeit, sich praktische Erfahrungen mit der Programmiersprache Python aneignete, erfolgte die Entwicklung des Programms in Python. Des Weiteren trugen die Java-Vorkenntnisse aus dem OOP-Modul des ersten Semesters zum leichteren Erlernen der Python-Programmiersprache bei. Während der Implementierung gingen auch mehrere Herausforderungen einher, die innerhalb des Kapitels Aufbau und Entwicklung ausführlich beschrieben sind. Die Zusammenarbeit im Team war stets gut und produktiv und sorgte dafür, das Projekt erfolgreich abzuschließen. Darüber hinaus förderte das Werkstück A das analytische sowie logische Denken, indem das Team versuchte, Probleme zu erkennen, in ihre Einzelteile zu zerlegen und dafür eigenständig strukturierte Lösungen zu entwickeln.

7. Ausblick

Das Auslesen der Bildschirmauflösung funktioniert im implementierten Programm nur unter Windows. Für ein erfolgreiches Ausführen unseres Spiels unter OSX, musste das Team die lokale Systemauflösung ‘manuell’ eintragen. Hier besteht ein Verbesserungspotenzial, indem das Spiel durch eine betriebssystemspezifische API automatisch erkennt auf welcher Plattform das Programm auszuführen ist. Des Weiteren könnte der Spieler in der graphischen Ausgabe statt dem Rect-Objekt durch eine Spielfigur mit animierter Richtungsbewegung ersetzt werden, um die graphische Ausgabe interessanter zu gestalten. Darüber hinaus könnten zukunftsorientiert akustische Signale zum einen das Laufen gegen eine Wand und zum anderen das Erreichen des Ziels signalisieren. Zu guter Letzt wäre eine weiteres Verbesserungspotenzial, die Eingabe der Achsenwerte nur über die GUI zu ermöglichen und zusätzlich das Programm als eine Web-App entwickeln, um dem Benutzer die Möglichkeit zu geben, das Maze-Spiel auf dem Handy zu spielen.

VI. Literaturverzeichnis

3D Monster (2020): Entwicklung. [Online] https://de.wikipedia.org/wiki/3D_Monster_Maze [Stand: 20.04.2020].

Box-drawing character (2021): Unicodes. [Online] https://en.wikipedia.org/wiki/Box-drawing_character [Stand: 29.05.2021].

Grundlegendes zur Stapelimplementierung in Python (2021): Operationen im Stapel. [Online] https://geekflare.com/de/python-stack-implementation/ [Abrufdatum: 22.06.2021].

Holzer, Raphael (2019): Welcome to making apps with Pygame. [Online] https://pygame.readthedocs.io/en/latest/ [Abrufdatum: 04.06.2021].

Holzer, Raphael (2020): Pygame Tutorial, Work with rectangles. [Online] https://pygame.readthedocs.io/en/latest/rect/rect.html [Stand: 2019].

Maze Generation (2015): Algorithm Recap, recursive backtracker. [Online] http://weblog.jamisbuck.org/2011/2/7/maze-generation-algorithm-recap [Stand: 02.2015].

Maze generation algorithm (2021): Iterative Implementation-Pseudocode [Online] https://en.wikipedia.org/wiki/Maze_generation_algorithm#:~:text=Iterative%20implementation%5Bedit%5D [Stand: 18.06.2021].

Maze generation algorithm (2021): Randomized depth-first search, Iterative Implementation. [Online] https://en.wikipedia.org/wiki/Maze_generation_algorithm#:~:text=exit%2C%20are%20removed.-,Randomized%20depth-first%20search,-%5Bedit%5D [Stand: 13.05.2021].

Maze generation algorithm (2021): Rekursive Implementation. [Online] https://en.wikipedia.org/wiki/Maze_generation_algorithm#:~:text=branch%20before%20backtracking.-,Recursive%20implementation,-%5Bedit%5D 5D [Stand: 18.06.2021].

McGugan, Will (2007): Beginning Game Development with Python and Pygame. New York: Springer.

McManus, Sean (2018): Mission Python, Code a space adventure Game. San Francisco.

Pratzner, Axel (2021): Pygame Library nutzen zum Spiele programmieren in Python. [Online] https://www.python-lernen.de/pygame-tutorial.htm [Stand: 25.03.2021].

Pullen, Walter D. (2021): Perfect Maze Creation Algorithms. [Online] http://www.astrolog.org/labyrnth/algrithm.htm#:~:text=and%20loop%20removers.-,Perfect%20Maze%20Creation%20Algorithms,-There%20are%20a [Stand: 22.03.2021].

Sweigart, Al. (2017): Invent your own Computer Games with Python, 4th Edition. San Francisco.

Williams, Lauren K. (2019): Recursive Backtracking Maze Generation. [Online] https://integral-domain.org/lwilliams/Applets/algorithms/backtrackingmaze.php [Stand: Juli 2019].

VII. Anhang

```
1    # coding=utf-8
2    import pygame
3    import random
4    import typing
5    from konstanten import *
6
7    class Koordinate(object):
8        def __init__(self, y: int, x: int, kantenlaenge: int):
9            self.y: int = y  # Vertikale   - Achse
10           self.x: int = x  # Horizontale - Achse
11           self.laenge: int = kantenlaenge
12           self.kanten: dict = {}
13           self.marker = ""  # type:  Rect or str
14           self.markerColor = None
15           self.solutionMarker = None # type: Rect or None
16           self.solutionMarkerColor = None
17           #                    right           down         left           up
18           self.neighbours =[(y, x+1, "v"), (y+1, x, "h"), (y, x-1, "v"), (y-1, x, "h")]
19           self.isVisited: bool = False
20
21       def __repr__(self) -> str:
22           return "{},{}".format(self.y, self.x)
23
24       @property
25       def getKoordinatenKantenDaten(self) -> str:
26           return f" |[{self}]←{self.kanten}|"
27
28       @property
29       def rect(self) -> pygame.Rect:
30           x = FENSTER_RAND_ABSTAND + self.x * self.laenge
31           y = FENSTER_RAND_ABSTAND + self.y * self.laenge
32           width = self.laenge
33           height = self.laenge
34           return Rect(x, y, width, height)
35
36
37   class Maze(object):
38       def __init__(self, yAchse: int, xAchse: int, kantenlaenge) -> None:
39           self.yAchse, self.xAchse = yAchse, xAchse
40           self.isPrintMarker = True
41           self.labyrinth = [[Koordinate(row, colmn, kantenlaenge)
42                              for colmn in range(self.xAchse + 1)] for row in range(self.yAchse + 1)]
43
44           self.wallStartChar = {'1111': '┼',
45                                 '0111': '┤', '1011': '┴', '1101': '├', '1110': '┬',
46                                 '0011': '┘', '1001': '└', '1100': '┌', '0110': '┐',
47                                 '0101': '│', '1010': '─',
48                                 '0001': '╵', '1000': '-', '0100': '╷', '0010': '-',
49                                 '0000': ' '}
50
51       def isValid_and_isNotVisited(self, y: int, x: int) -> bool:
52           return (0 <= y < self.yAchse) and (0 <= x < self.xAchse) \
53                  and not self.labyrinth[y][x].isVisited
54
55       def isValid(self, y: int, x: int) -> bool:
56           return 0 <= y < self.yAchse and 0 <= x < self.xAchse
57
```

```
58      def __repr__(self) -> str:
59          ausgabe = " "
60          for y in range(self.yAchse + 1):
61              for x in range(self.xAchse):       # Zeile: H-Kante-Zeile
62                  if "h" in self.labyrinth[y][x].kanten:
63                      ausgabe += '{}━━━━━━━'.format(self.wallStartChar[self._getZeichenCode(y, x)])
64                  else:
65                      ausgabe += '{}       '.format(self.wallStartChar[self._getZeichenCode(y, x)])
66              ausgabe += '{}\n '.format(self.wallStartChar[self._getZeichenCode(y, self.xAchse)])
67
68              for x in range(self.xAchse + 1):  # Zeile: V-Kante-Zeile
69                  if "v" in self.labyrinth[y][x].kanten:
70                      if self.labyrinth[y][x].marker and self.isPrintMarker:
71                          ausgabe += '┃{}'.format(self.labyrinth[y][x].marker)
72                      elif self.labyrinth[y][
73                          x].solutionMarker and not self.isPrintMarker:
74                          ausgabe += '┃ ██ '
75                      else:
76                          ausgabe += '┃    '
77                  else:
78                      if self.labyrinth[y][x].marker and self.isPrintMarker:
79                          ausgabe += ' {}'.format(self.labyrinth[y][x].marker)
80                      elif self.labyrinth[y][
81                          x].solutionMarker and not self.isPrintMarker:
82                          ausgabe += '  ██ '
83                      else:
84                          ausgabe += '     '
85              ausgabe += '\n '
86
87          ausgabe = ausgabe[:(-5 * self.xAchse) - 10] + '┛\n'
88          return ausgabe
89
90      def _getZeichenCode(self, ky, kx) -> str:
91          zeichenCode =  "1" if 'h' in self.labyrinth[ky][kx].kanten else "0"
92          zeichenCode += "1" if 'v' in self.labyrinth[ky][kx].kanten else "0"
93          zeichenCode += "1" if (self.isValid(ky, kx - 1) and 'h' in self.labyrinth[ky][kx - 1].kanten) \
94                     else "0"
95          zeichenCode += "1" if (self.isValid(ky - 1, kx) and 'v' in self.labyrinth[ky - 1][kx].kanten) \
96                     else "0"
97         #       ┏→ Rand unten              ┏→ 1.EckeUntenLinks
98          if (ky == self.yAchse and not kx == 0) and (zeichenCode == '1000' or zeichenCode == '1001'):
99          #          ┏→'━━' if ↓    GLEICH ↓    ┏→'-'  ┏ELSE   ┏→'┻'      ┗→'━━'              ┗→'┗',
100             return '1010' if zeichenCode == '1000' else '1011'
101         #       ┏→ Rand rechts               ┏→ 4.EckeObenRechts
102         if (kx == self.xAchse and not ky == 0) and (zeichenCode == '0100' or zeichenCode == '0110'):
103                  #   ┏→'┃' if     ↓ GLEICH ↓   ┏→'╻'   ┏ELSE   ┏→'┫'       ┗→'╻'              ┗→'┓'
104             return '0101' if zeichenCode == '0100' else '0111'
105
106         return zeichenCode
107
108
```

```
109 class Player(object):
110     def __init__(self, y_achse:int, x_achse:int, spanningTree: dict):
111         self.currentKy, self.zielKy, self.currentKx, self.zielKx = 0, 0, 0, 0
112 
113         while (self.currentKy, self.currentKx) == (self.zielKy, self.zielKx):
114             self.currentKy, self.zielKy = random.randint(0, y_achse - 1), random.randint(0, y_achse - 1)
115             self.currentKx, self.zielKx = random.randint(0, x_achse - 1), random.randint(0, x_achse - 1)
116 
117         self.spanningTree = spanningTree
118         #self.directions = {'right': (0, 1), 'left': (0, -1), 'down': (1, 0), 'up': (-1, 0)}
119 
120     def isDirectionValid(self, yvon:int, xvon:int, ynach:int, xnach:int) -> bool:
121         return [ynach, xnach] in self.spanningTree[(yvon, xvon)]
122 
123     def setPos(self, ky: int, kx: int):
124         self.currentKy, self.currentKx = ky, kx
125 
126     def getPos(self) -> typing.Tuple[int, int]:
127         return self.currentKy, self.currentKx
128 
129 
130 class Stack(object):
131     def __init__(self):
132         self.liste = []  # type: list
133 
134     def push(self, koordinate: Koordinate):
135         self.liste.append(koordinate)
136 
137     def popp(self) -> Koordinate:
138         return self.liste.pop(-1)
139 
140     @property
141     def size(self) -> int:
142         return len(self.liste)
143 
144     def isNotEmpty(self) -> bool:
145         return self.size != 0
146 
147     def __repr__(self) -> str:
148         ausgabe = ""
149         for koordinate in self.liste: ausgabe += f"({koordinate}), "
150         return ausgabe
```

Listing 1: model.py

```
1    import copy
2    from typing import *
3    from model import *
4
5    class MazeGenerator(object):
6        def __init__(self, y_Achse: int, x_Achse: int, kantenlaenge: int):
7            self.y_Achse: int = y_Achse
8            self.x_Achse: int = x_Achse
9            self.kantenlaenge: int = kantenlaenge
10           self.startKy: int  = random.randint(0, y_Achse - 1)
11           self.startKx: int  = random.randint(0, x_Achse - 1)
12           self.maze: Maze = Maze(self.y_Achse, self.x_Achse, kantenlaenge)
13           self.labyrinth: List[List[Koordinate]] = self.maze.labyrinth
14           self.spanning3: dict = {}
15           self.stack: Stack = Stack()
16           self.createWalls()
17           self.createMaze(self.labyrinth[self.startKy][self.startKx])
18
19       def createWalls(self) -> None:
20           for y in range(self.y_Achse + 1):
21               for x in range(self.x_Achse + 1):
22                   vh_x = FENSTER_RAND_ABSTAND + (x * self.kantenlaenge)
23                   vh_y = FENSTER_RAND_ABSTAND + (y * self.kantenlaenge)
24                   if x < self.x_Achse:  # Bildung der horizontalen Kantendimension
25                       self.labyrinth[y][x].kanten['h'] = Rect(vh_x, vh_y, self.kantenlaenge, HOEHE)
26                   if y < self.y_Achse:  # Bildung der vertikalen Kantendimension
27                       self.labyrinth[y][x].kanten['v'] = Rect(vh_x, vh_y, HOEHE,self.kantenlaenge)
28
29       def createMaze(self, aktuelZel: Koordinate) -> None:
30           aktuelZel.isVisited = True  # 1.
31           self.stack.push(aktuelZel)  # 1.
32           while self.stack.isNotEmpty():  # 2.
33               aktuelZel = self.stack.popp()  # type: Koordinate       # 2.1
34
35               if (aktuelZel.y, aktuelZel.x) not in self.spanning3:
36                   self.spanning3[(aktuelZel.y, aktuelZel.x)] = []
37
38               if aktuelZel.neighbours:  # 2.2
39                   self.stack.push(aktuelZel)  # 2.2.1
40                   nextCell_y, nextCell_x, kantenTyp =random.choice(aktuelZel.neighbours)
41                   aktuelZel.neighbours.remove((nextCell_y, nextCell_x, kantenTyp))
42
43                   if self.maze.isValid_and_isNotVisited(nextCell_y, nextCell_x): # 2.2
44                       nxtZel = self.labyrinth[nextCell_y][nextCell_x]  # 2.2 & 2.2.2
45
46                       if (nxtZel.y + nxtZel.x - aktuelZel.y - aktuelZel.x) > 0:
47                           self.deleteWall(nxtZel, kantenTyp)  # 2.2.3
48                       else:
49                           self.deleteWall(aktuelZel, kantenTyp)  # 2.2.3
50
51                       self.spanning3[(aktuelZel.y, aktuelZel.x)].append([nxtZel.y, nxtZel.x])
52
53                       if (nxtZel.y, nxtZel.x) in self.spanning3:
54                           self.spanning3[nxtZel.y,nxtZel.x].append([aktuelZel.y, aktuelZel.x])
55                       else:
56                           self.spanning3[nxtZel.y,nxtZel.x] = [[aktuelZel.y, aktuelZel.x]]
57
58                       nxtZel.isVisited = True  # 2.2.4
59                       self.stack.push(nxtZel)  # 2.2.4
60
61       def deleteWall(self, k: Koordinate, wandTyp: str) -> None:
62          del self.maze.labyrinth[k.y][k.x].kanten[wandTyp]
63
64       def getKoordinatenData(self) -> str:
65           ausgabe: str = ""
66           for y in range(self.y_Achse):
67               for x in range(self.x_Achse): ausgabe +=f"{self.labyrinth[y][x].getKoordinatenKantenDaten} "
69               ausgabe += "\n"
70           return ausgabe
71
```

```
72
73  class PathFinder(object):
74      def __init__(self, mazerator:MazeGenerator, player:Player, isDoPathFinder=True):
75          self.stack: Stack = Stack()  # der Lösungspfad
76          if isDoPathFinder:
77              self.validPath: dict = copy.deepcopy(mazerator.spanning3)
78              self.labyrinth: List[List[Koordinate]] = mazerator.labyrinth
79              self.player: Player = player
80              self.findPath()
81              self.solutionPath2Labyrinth()
82
83      def findPath(self) -> None:
84          nextCell_y, nextCell_x = -1, -1
85          self.stack.push(self.labyrinth[self.player.currentKy][self.player.currentKx])
86
87          while (nextCell_y, nextCell_x) != (self.player.zielKy, self.player.zielKx):
88              currentCell = self.stack.popp()
89              validPathList = self.validPath[(currentCell.y, currentCell.x)]
90              if validPathList:
91                  self.stack.push(currentCell)
92
93                  nextCell_y, nextCell_x = random.choice(validPathList)
94                  self.validPath[(currentCell.y, currentCell.x)].remove([nextCell_y,nextCell_x])
95                  self.validPath[(nextCell_y, nextCell_x)].remove([currentCell.y,currentCell.x])
96
97                  self.stack.push(self.labyrinth[nextCell_y][nextCell_x])
98
99      def solutionPath2Labyrinth(self) -> None:
100         for cell in self.stack.liste:
101             cell.solutionMarker = self.calculateRect(cell)
102             cell.solutionMarkerColor = SOLUTIONPATHCOLOR
103
104     def resetMarker(self) -> None:
105         for cell in self.stack.liste:
106             cell.solutionMarker = None
107
108     @staticmethod
109     def calculateRect(k: Koordinate):
110         x = FENSTER_RAND_ABSTAND + k.x * k.laenge + (k.laenge / 4)
111         y = FENSTER_RAND_ABSTAND + k.y * k.laenge + (k.laenge / 4)
112         width = k.laenge - (k.laenge / 2)
113         height = k.laenge - (k.laenge / 2)
114         return Rect(x, y, width, height)
115
```

Listing 2: algo.py

```
1    # coding=utf-8
2    import argparse
3    import copy
4    import pygame
5    import re
6    import sys
7    import time
8    from typing import *
9
10   from algo import MazeGenerator, PathFinder
11   from konstanten import *
12   from model import Player, Koordinate
13
14   class MazeSpiel:
15       def __init__(self, y_Achse: int, x_Achse: int, maze_generator: MazeGenerator,
16                    player: Player, screentype: int):
17           # Pygame init, Anzeigemodus m. ggf. Größenberechnung, Bildschirmfläche, FPS
18           pygame.init()
19           self.flags = screentype                                  # type: int
20           self.screenSize = SCREENSIZE                             # type: Tuple[int, int]
21           if self.flags == RESIZABLE:
22               self.screenSize =(2 * FENSTER_RAND_ABSTAND + x_Achse * LAENGE + HOEHE),\
23                                (2 * FENSTER_RAND_ABSTAND + y_Achse * LAENGE + HOEHE)
24           self.screen = pygame.display.set_mode(self.screenSize, self.flags,32)
25           pygame.display.set_caption('MazeGame')
26           self.framesPerSecond = x_Achse if x_Achse > y_Achse else y_Achse     # type: int
27           # Achsenwerte
28           self.xAchse, self.yAchse = x_Achse, y_Achse              # type: int, int
29
30           # Labyrinth, Wandfarbe, Spannbaum (spanning3, sp3)
31           self.mazerator = maze_generator                          # type: MazeGenerator
32           self.labyrinth = maze_generator.labyrinth                # type: List[List[Koordinate]]
33           self.kantenfarbe = (WALLCOLOR_1, WALLCOLOR_1)
34           self.spanning3_1stKey = maze_generator.startKy, maze_generator.startKx  # type: Tuple[int, int]
35           self.sp3_ykey, self.sp3_xkey, self.sp3copy = 0,0,None   # type: int,int, None
36
37           # Player, Index-Modifikatoren für Spielerbewegungsrichtung, pathFinder init
38           self.player     = player                                 # type: Player
39           self.startKy, self.startKx = self.player.currentKy, self.player.currentKx    # type: int, int
40           self.directions = {'right':(0, 1), 'left':(0, -1), 'down':(1, 0), 'up':(-1, 0)}
41           self.pathFinder = None                                   # type: None or PathFinder
42
43           # Boolean-Switch, Counter-Variablen und solutionSize (Lösungspfadlänge) init
44           self.isShowMenu, self.isRunning = True, True             # type: bool, bool
45           self.isShowSolutionPath, self.isShowSpaning3, self.is2farbig = False, False, False
46           self.totalMoves, self.invalidMoves  = 0, 0               # type: int, int
47           self.solutionSize, self.gTasteCount = 0, 0               # type: int, int
48
49           # Menüvorbereitung, Überschreiben der entsprechenden Marker der jeweiligen Koordinaten-Instanzen
50           self.menuText_ImageList = self.initMenu()
51           self.markStart_Ziel(self.player.currentKy, self.player.currentKx, STARTCOLER)
52           self.markStart_Ziel(self.player.zielKy, self.player.zielKx, ZIELCOLER)
53
```

```
54      @staticmethod
55      def initMenu():                                                 # Menü-Init
56          font_24 = pygame.font.SysFont('consolas.ttf', 24)
57          font_20 = pygame.font.SysFont('consolas.ttf', 20)
58          titleColor, textColor, infoColor = ROT, GRUENNEON, BLAUNEON
59          text_Images = [ font_24.render(6*" "+'M a z e G a m e  [ m ] E N Ü      ',
60                                                                  True,titleColor),
61              font_24.render('   Pfeiltasten: < ^  > v ',True, textColor),
62              font_24.render(
63                  '                                oder mit: a x d s ', True, textColor),
64              font_24.render(
65                  '                  Lösungspfad: F1          ',     True, textColor),
66              font_24.render('                Spanning Tree: Enter    ',
67                                                                   True, textColor),
68              font_24.render(' Start-Ziel Neuvergabe: g             ',True, textColor),
69              font_24.render('     Wandfarben-Switch: k             ',True, textColor),
70              font_24.render('               Spiel beenden: q, Esc    ',
71                                                                   True, textColor),
72              font_20.render('     Start, Ziel? Evtl. hinter Menü?          ',
73                                                                   True, infoColor)]
74          return text_Images
75
76      def markStart_Ziel(self, ky: int, kx: int, farbe: Tuple[int, int, int]):
77          self.labyrinth[ky][kx].marker = self.labyrinth[ky][kx].rect
78          self.labyrinth[ky][kx].markerColor = farbe
79
80      def run(self):                                              # Hauptschleife Spiel
81          isSuccess, playerTimer = False, time.time()
82          clock = pygame.time.Clock()                                  # FPS-Begrenzer
83          while self.isRunning:
84              self.screen.fill(BGCOLOR)                    # reset screen auf BGCOLOR
85
86              for ereignis in pygame.event.get():              # Benutzerinteraktion
87                  self.do_pygameEvents(ereignis)
88
89              self.do_drawLabyrinth()                            # Zeichne Labyrinth
90
91              if self.isShowMenu:                                  # Zeige Menü an
92                  self.do_drawMenu()
93
94              if self.isShowSpaning3:  # Nutzt while Schleife zum spanning3 animieren
95                  self.do_showSpanningTree(clock)
96
97              pygame.display.flip()         # Gebe gezeichnete Objekte auf screen aus
98              clock.tick(self.framesPerSecond)
99                                  # Sobald Spieler Ende erreicht wird Spiel beendet.
100             if self.player.getPos() == (self.player.zielKy, self.player.zielKx):
101                 playerTimer = time.time() - playerTimer
102                 isSuccess = True
103                 self.isRunning = False
104
105         self.do_printGameMetrics(isSuccess, playerTimer)        # Gebe Auswertung aus
106         pygame.quit()                                              # Beendet Pygame
107
```

```
108     def do_pygameEvents(self, ereignis: pygame.event.Event):  # B E N U T Z E R I N T E R A K T I O N
109         if ereignis.type == QUIT:
110             self.isRunning = False   # A K T I O N: Fenster "X" via Maus zum SpielBeenden
111         elif ereignis.type == pygame.KEYDOWN:
112             # A K T I O N: Alternatives vorzeitiges Beenden des Spiels Spielsteuerung
113             if ereignis.key == pygame.K_q or ereignis.key == pygame.K_ESCAPE:  self.isRunning = False
114                                 # A K T I O N:   S P I E L S T E U E R U N G
115             elif ereignis.key == pygame.K_RIGHT or ereignis.key == pygame.K_d: self.on_keyEvent('right')
116
117             elif ereignis.key == pygame.K_DOWN or ereignis.key == pygame.K_s:  self.on_keyEvent('down')
118
119             elif ereignis.key == pygame.K_LEFT or ereignis.key == pygame.K_a:  self.on_keyEvent('left')
120
121             elif ereignis.key == pygame.K_UP or ereignis.key == pygame.K_w:    self.on_keyEvent('up')
122                                             # A K T I O N: Lösungspfad
123             elif ereignis.key == pygame.K_F1:
124                 self.isShowSolutionPath = not self.isShowSolutionPath
125                 self.isShowSpaning3 = False
126                 self.on_keyEvent_F1()
127                                 # A K T I O N: Neuvergabe Start und Ziel zufällig
128             elif ereignis.key == pygame.K_g:
129                 self.on_keyEvent_g()
130                 self.gTasteCount += 1
131                                         # A K T I O N: Animation Spannbaum
132             elif ereignis.key == pygame.K_RETURN:
133                 if    self.pathFinder:  self.pathFinder.resetMarker()
134                 else: self.pathFinder = PathFinder(self.mazerator,self.player,False)
135
136                 self.sp3copy = copy.deepcopy(self.mazerator.spanning3)
137                 self.sp3_ykey, self.sp3_xkey = self.spanning3_1stKey
138                 self.isShowSpaning3 = not self.isShowSpaning3
139                                 # A K T I O N: Switch Kantenfarbe
140             elif ereignis.key == pygame.K_k: self.on_keyEvent_k()   # ↓Aktion↓: spanning3-Konsolenausgabe
141             elif ereignis.key == pygame.K_t: print("\nSpannbaum-Daten:\n",self.mazerator.spanning3,"\n")
142             elif ereignis.key == pygame.K_m: self.isShowMenu = not self.isShowMenu
143             if   ereignis.key != pygame.K_m:  self.isShowMenu = False
144
145     def do_drawLabyrinth(self):                                                 # Zeichne Labyrinth
146         for row in range(self.yAchse + 1):                        # +1? Zur Anzeige des unteren Rands
147             for column in range(self.xAchse + 1):                                  # +1? rechter Rand
148
149                 k_instanz = self.labyrinth[row][column]
150                 if k_instanz.marker:                      # Feld für Spielverlauf, akt. Position, Ziel
151                     pygame.draw.rect(self.screen, k_instanz.markerColor, k_instanz.marker)
152                                                                 # für Lösungspfad oder Spannbaum
153                 if (self.isShowSolutionPath or self.isShowSpaning3) and k_instanz.solutionMarker:
154                     pygame.draw.rect(self.screen,k_instanz.solutionMarkerColor, k_instanz.solutionMarker)
155
156                 if 'h' in k_instanz.kanten:                                      # horizontale Wand
157                     pygame.draw.rect(self.screen, self.kantenfarbe[column % 2], k_instanz.kanten['h'])
158
159                 if 'v' in k_instanz.kanten:                                        # vertikale Wand
160                     pygame.draw.rect(self.screen, self.kantenfarbe[column % 2], k_instanz.kanten['v'])
161
162
163     def do_drawMenu(self):                                              # Zeige Menü
164         x, y = 30, 40
165         for textImage in self.menuText_ImageList:
166             pygame.draw.rect(self.screen,BGCOLOR,Rect(x,y,textImage.get_rect().w,textImage.get_rect().h))
167             self.screen.blit(textImage, (x, y))
168             y += 16
169
```

```
170     def do_showSpanningTree(self, clock: pygame.time.Clock()): # Animation spaning3
171
172         if self.sp3copy[self.sp3_ykey, self.sp3_xkey]:
173             k = self.labyrinth[self.sp3_ykey][self.sp3_xkey]
174
175             if not k.solutionMarker:
176                 k.solutionMarker = k.rect
177                 self.pathFinder.stack.push(k)
178                 k.solutionMarkerColor = GENERATOR_COLOR
179             else:
180                 k.solutionMarkerColor = BACKTRACKER_COLOR
181
182             self.sp3_ykey, self.sp3_xkey = self.sp3copy[self.sp3_ykey, self.sp3_xkey].pop()
183
184             pygame.display.update(pygame.draw.rect(self.screen, k.solutionMarkerColor, k.solutionMarker))
185             if 'h' in k.kanten:  # horizontale Wand
186                 pygame.display.update(pygame.draw.rect(self.screen, self.kantenfarbe[0], k.kanten['h']))
187             if 'v' in k.kanten:  # vertikale Wand
188                 pygame.display.update(pygame.draw.rect(self.screen, self.kantenfarbe[0], k.kanten['v']))
189             clock.tick(self.framesPerSecond)
190
191     def do_printGameMetrics(self, isSuccess: bool, playerTimer: float):  # Auswertung
192         self.player.setPos(self.startKy, self.startKx)
193         self.isShowSolutionPath = True  # ← damit Lösungspfad in
194         self.on_keyEvent_F1()           # on_keyEvent_F1 überhaupt berechnet werden kann
195
196         validMove    = self.totalMoves - self.invalidMoves
197         validquote   = 0 if validMove == 0 else validMove * 100 / self.totalMoves
198         invalidquote = 0 if self.invalidMoves == 0 else self.invalidMoves * 100 / self.totalMoves
199         totalquote   = 0 if not isSuccess else self.solutionSize * 100 / self.totalMoves
200
201         if isSuccess:
202             print(AUSWERTUNG_MSG.format('{:6.2f}'.format(playerTimer), "erfolgreich", self.gTasteCount))
203         else:
204             playerTimer = time.time() - playerTimer
205             print(AUSWERTUNG_MSG.format('{:6.2f}'.format(playerTimer),"vorzeitig", self.gTasteCount))
206
207         print(SOLUTION_MSG.format(self.totalMoves, validMove,'{:5.2f}'.format(validquote),
208                                         self.invalidMoves,'{:5.2f}'.format(invalidquote),
209                                         self.solutionSize,'{:5.2f}'.format(totalquote)))
210
211     def on_keyEvent(self, direction: str): # Prüfung Bewegungsrichtung des Spielers
212         self.isShowSpaning3, self.isShowSolutionPath = False, False
213         self.totalMoves += 1
214         mody, modx = self.directions[direction]
215         yvon, xvon = self.player.getPos()
216         ynach = yvon + mody
217         xnach = xvon + modx
218
219         if self.player.isDirectionValid(yvon, xvon, ynach, xnach):
220             self.labyrinth[yvon][xvon].markerColor = PLAYERPATHCOLOR
221             self.labyrinth[ynach][xnach].marker = self.labyrinth[ynach][xnach].rect
222             self.labyrinth[ynach][xnach].markerColor = VALIDMOVECOLOR
223             self.player.setPos(ynach, xnach)
224         else:
225             self.labyrinth[yvon][xvon].markerColor = INVALIDMOVECOLOR
226             self.invalidMoves += 1
227
228     def on_keyEvent_F1(self):                               # Berechne Lösungspfad
229         if self.isShowSolutionPath:
230             if self.pathFinder:                  # reset alten Lösungspfad
231                 self.pathFinder.resetMarker()    # oder spanning3 in solutionMarker
232             self.pathFinder = PathFinder(self.mazerator, self.player)
233             self.solutionSize = self.pathFinder.stack.size - 1
234
```

```
235      def on_keyEvent_g(self):                                # Neuvergabe Start / Ziel
236          self.isShowSpaning3 = False
237          for zeile in range(self.yAchse):
238              for spalte in range(self.xAchse):
239                  self.mazerator.labyrinth[zeile][spalte].marker = None
240                  self.mazerator.labyrinth[zeile][spalte].solutionMarker = None
241
242          self.player = Player(self.yAchse, self.xAchse, self.mazerator.spanning3)
243          self.startKy, self.startKx = self.player.currentKy, self.player.currentKx
244          self.markStart_Ziel(self.player.currentKy, self.player.currentKx,STARTCOLER)
245          self.markStart_Ziel(self.player.zielKy, self.player.zielKx, ZIELCOLER)
246          self.on_keyEvent_F1()
247
248      def on_keyEvent_k(self):
249          self.is2farbig = not self.is2farbig
250          if self.is2farbig: self.kantenfarbe = (WALLCOLOR_1, WALLCOLOR_2)
251          else:              self.kantenfarbe = (WALLCOLOR_1, WALLCOLOR_1)
252
253
254  class Konsole(object):
255      def __init__(self, yAchse: int = 10, xAchse: int = 10):
256          self.yAchse: int = yAchse
257          self.xAchse: int = xAchse
258          self.kantenlaenge: int = LAENGE
259          self.screentype: int = SCREENTYPE
260          self.mazerator = None  # type: None or  MazeGenerator
261          self.running, self.debug = True, False  # type: bool, bool
262
263      def run(self):                                          # Hauptschleife Konsole
264          while self.running:
265              # M E N U  A U S G A B E  →  A C H S E N  I N P U T
266              # → V A L I D I E R U N G → Wandlängenberechnung für die graph. Ausgabe
267              self.setXYachsen()
268              if not self.running:
269                  break
270
271              # L A B Y R I N T H  E R S T E L U N G
272              startTimeToCreate = time.time()
273              self.mazerator = MazeGenerator(self.yAchse, self.xAchse, self.kantenlaenge)
274
275              # Zeitdauer zum Generieren und Auswahl der Zeiteinheit
276              timeToCreateMaze: float = time.time() - startTimeToCreate
277              mazeCreationTime: float = (timeToCreateMaze * 1000) if (timeToCreateMaze < 1) \
278                                    else timeToCreateMaze
279              mazeCreationTimeUnit: str = "Millisekunden" if (timeToCreateMaze < 1) \
280                                     else "Sekunden"
281
282              # P L A Y E R  I N S T A N Z I E R U N G + Start/Ziel Markierung
283              player = Player(self.yAchse, self.xAchse, self.mazerator.spanning3)
284              labyrinth = self.mazerator.labyrinth
285              labyrinth[player.currentKy][player.currentKx].marker = "PLAY"
286              labyrinth[player.zielKy][player.zielKx].marker = "END "
287
288              # L a b y r i n t h - A u s g a b e  a l s  D a t e n
289              if self.debug: print(f"{self.mazerator.getKoordinatenData()}\n\n")
290
291              # K O N S O L E N - A U S G A B E - L A B Y R I N T H
292              print(f" Labyrinth {self.yAchse, self.xAchse}:\n{self.mazerator.maze}")
293              # Gesamtzeitdauer inkl. Berechnung des Ausgabestrings sowie Ausgabe
294              totalTime = time.time() - startTimeToCreate
295              timeSinceCreation = (totalTime * 1000) if (totalTime < 1) else totalTime
296              sinceCreationTimeUnit ="Millisekunden" if (totalTime < 1) else "Sekunden"
297
298              # I n f o  A u s g a b e
299              print(f" Labyrinth {self.yAchse, self.xAchse} generiert in:"
300                    f"{'{:6.2f}'.format(mazeCreationTime)} {mazeCreationTimeUnit}\n"
301                    f" Gesamtdauer inkl. Ausgabe in der Konsole:"
302                    f"{'{:6.2f}'.format(timeSinceCreation)} {sinceCreationTimeUnit}\n")
303
```

```
304             # E R S T E L U N G  /  S T A R T  D E S  P I E L S
305             MazeSpiel(self.yAchse, self.xAchse, self.mazerator, player,
306                       self.screentype).run()
307
308             # K O N S O L E N - A U S G A B E - L A B Y R I N T H  mit Lösungspfad
309             self.mazerator.maze.isPrintMarker = False
310             print(f" Labyrinth {self.yAchse, self.xAchse}:\n"
311                   f"{self.mazerator.maze}")
312
313         print("\nProgramm wurde beendet. Vielen Dank fürs Spielen.")
314
315     def setXYachsen(self):                                  # Achsenwert Eingabe
316         while True:
317             try:
318                 userinput = input(MENU).lower().strip()
319                 ystring, xstring = "",""
320                 if "q" in userinput:
321                     self.running = False
322                     break
323                 elif "v" in userinput:
324                     isAxisValid, self.yAchse, self.xAchse,self.kantenlaenge, self.screentype = \
325                         get_isAxisValid_kantenlen_scrntype(self.yAchse, self.xAchse)
326                     if isAxisValid:
327                         break
328                 elif "d" in userinput:
329                     self.debug = not self.debug
330                     print("\n", 12 * " ", "Debug Info-Ausgabe? ", self.debug)
331                 elif ("x" in userinput) or ("*" in userinput) or (" " in userinput):
332                     xstring, ystring = re.split('[x *]',userinput)
333                 else:
334                     xstring = input(" Bitte x-Achse eingeben (horizontale):")
335                     ystring = input(" Bitte y-Achse eingeben (vertikale):")
336                 try:
337                     self.yAchse, self.xAchse = int(ystring), int(xstring)
338                     isAxisValid, self.yAchse, self.xAchse, self.kantenlaenge, self.screentype =\
339                         get_isAxisValid_kantenlen_scrntype(self.yAchse, self.xAchse)
340                     if isAxisValid:
341                         break
342                 except ValueError:
343                     if "d" not in userinput:
344                         print(WRONG_VALUE_ERRMSG)
345                 except TypeError:
346                     pass
347             except ValueError:
348                 print(TOO_MANY_VALUE_ERRMSG)
349
350
```

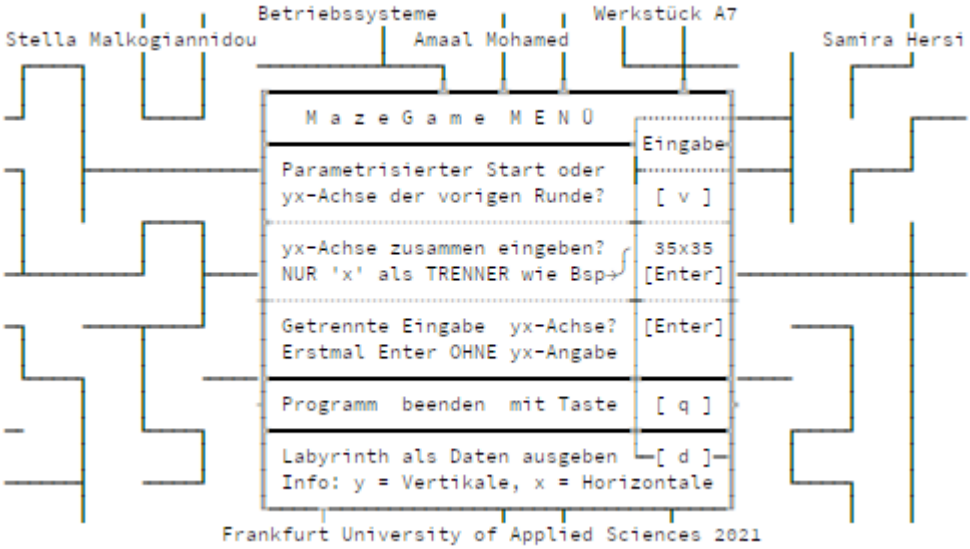
```
def get_isAxisValid_kantenlen_scrntype(yAchse, xAchse):  # Achsenwert Validierung
    kantenlaenge, minKantenLaenge = LAENGE, KANTENLAENGE_MINIMUM  # type: int, int
    x_screenTotal, y_screenTotal = SCREENSIZE  # type: int
    screentype = RESIZABLE  # type: int or None
    usableScreen_y = y_screenTotal - 2 * FENSTER_RAND_ABSTAND  # type: int
    usableScreen_x = x_screenTotal - 2 * FENSTER_RAND_ABSTAND  # type: int
    max_y_Achse = int(usableScreen_y / minKantenLaenge)  # type: int
    max_x_Achse = int(usableScreen_x / minKantenLaenge)  # type: int

    if yAchse < 10 or xAchse < 10: #) and (not yAchse == 6 and not xAchse == 9):
        if yAchse > 0 and xAchse > 0:
            print(AXIS_SMALLER_10_ERRMSG.replace("^", "{}").format(max_y_Achse, max_x_Achse))
        else:
            print(AXIS_SMALLER_0_ERRMSG)
        return False,yAchse,xAchse,None,None

    elif yAchse > max_y_Achse or xAchse > max_x_Achse:
        if yAchse > xAchse:
            temp = yAchse
            yAchse = xAchse  # vertauscht yAchsenWert mit xAchsenWert
            xAchse = temp
            if yAchse > max_y_Achse or xAchse > max_x_Achse:
                print(AXIS_TOO_BIG_ERRMSG.replace("^", "{}").format(max_y_Achse, max_x_Achse))
                return False,yAchse,xAchse,None,None
        else:
            print(AXIS_TOO_BIG_ERRMSG.replace("^", "{}").format(max_y_Achse, max_x_Achse))
            return False,yAchse,xAchse,None,None

    kantenlaenge_y, kantenlaenge_x = int(usableScreen_y / yAchse), int(usableScreen_x / xAchse)
    kantenlaenge = min(kantenlaenge_x,kantenlaenge_y) if min(kantenlaenge_x,kantenlaenge_y) <= 30 else 30
    x_maze2ScreenRatio = xAchse * kantenlaenge / usableScreen_x
    y_maze2ScreenRatio = yAchse * kantenlaenge / usableScreen_y
    #print(f"\n\n x_maze2ScreenRatio: {x_maze2ScreenRatio},\n y_maze2ScreenRatio: {y_maze2ScreenRatio}")
    screentype = FULLSCREEN if (y_maze2ScreenRatio > 0.85 or x_maze2ScreenRatio > 0.85) else RESIZABLE

    print(ISVALID_MSG)
    return True, yAchse, xAchse, kantenlaenge, screentype


def _get_args():
    """

    Quellenangabe:
    https://mkaz.blog/code/python-argparse-cookbook/
    YOUENS-CLARK, KEN (2020): Tiny Python Projects, Seite 32. New York: Manning Publications
    :return:
    :rtype:
    """
    parser = argparse.ArgumentParser( description = 'Parametrisierter Start des Maze-Spiels',
                                      formatter_class = argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument(   'axisValues', nargs = '*', type = int, help = AXIS_HELP_MSG)
    parser.add_argument('-x', '--xaxis', nargs = 1,   type = int, help = "x-Achsenwert als Integer")
    parser.add_argument('-y', '--yaxis', nargs = 1,   type = int, help = "y-Achsenwert als Integer")
    parser.add_argument('-gui', help = GUI_HELP_MSG, action = 'store_true')
    return parser.parse_args()

```

```
407 def main():
408     """
409     Prüft, ob Benutzer exakt 2 oder keine Parameter übergab.
410 
411     Übergabe von nur 1 Parameter und mehr als 2 Parameter führen zur Ausgabe einer
412     entsprechenden Fehlermeldung. Wenn Benutzer exakt 2 Parameter übergab, so wird das Programm
413     parametrisiert gestartet und falls keine Übergaben erfolgten, dann wird das Programm ohne
414     Übergabeparameter gestartet.
415     """
416     args = _get_args()
417     if args.gui and args.xaxis and args.yaxis:
418         isAxisValid, yAchse, xAchse, kantenlaenge,screentype \
419             = get_isAxisValid_kantenlen_scrntype(args.yaxis[0],args.xaxis[0])
420         if isAxisValid:
421             mazerator = MazeGenerator(yAchse, xAchse, kantenlaenge)
422             MazeSpiel(yAchse,xAchse,mazerator,Player(yAchse,xAchse,mazerator.spanning3),screentype).run()
423     else:
424         if args.xaxis and args.yaxis:         # Param.übg. via: -x 10 -y 11 oder --xaxis 10 --yaxis 11
425             print(PARAM_MSG.format( args.xaxis[0], args.yaxis[0] ))
426             Konsole(args.yaxis[0], args.xaxis[0]).run()
427         elif (args.xaxis and not args.yaxis) or (not args.xaxis and args.yaxis):
428             print( ONLY_1_PARAM_ERRMSG )
429         elif len(args.axisValues) == 2:                                    # Param.übg. via: 10 11
430             print(PARAM_MSG.format( args.axisValues[0], args.axisValues[1] ))
431             Konsole(args.axisValues[1], args.axisValues[0]).run()
432         elif len(args.axisValues)  > 2:  print(OVER_2_PARAM_ERRMSG)             # Zuviele Parameter
433         elif len(args.axisValues) == 1:  print(ONLY_1_PARAM_ERRMSG)   # Unvollständige Parameterübergabe
434         else:
435             print(NO_PARAM_MSG)                                           # Keine ParameterÜbergabe
436             Konsole().run()
437 
438 if __name__ == '__main__':
439     main()
440     sys.exit()
```

Listing 3: mazespiel.py

```
1    # coding=utf-8
2    import ctypes
3    from pygame.locals import *
4    # S I Z E S
5    _user32 = ctypes.windll.user32
6    # (x,y-Auflösung =      ┌──────────────→ x-Achse           ┌──────────────→  y-Achse
7    SCREENSIZE = _user32.GetSystemMetrics(0), _user32.GetSystemMetrics(1)
8    SCREENTYPE = RESIZABLE
9    LAENGE = 30
10   KANTENLAENGE_MINIMUM = 4
11   HOEHE = 1
12   FENSTER_RAND_ABSTAND = 2
13   #                                  C O L O R S
14   # (rot-Wert, ↓ , blau-Wert)
15   #     |     grün-Wert   |   Farbkodierung
16   #     └─ ──┐    ┌──┘ ┌──────────┘
17   BLACK = (0, 0, 0)  # (rot-Wert, grün-Wert, blau-Wert) Farbkodierung
18   BLAUNEON = (100, 200, 255)
19   GELBNEON = (255, 255, 0)
20   GRUENNEON = (0, 255, 0)
21   GRUEN_115 = (0, 115, 0)
22   GRUEN_180 = (0, 180, 0)
23   GRUEN_75 = (0, 75, 0)
24   PINK = (125, 0, 125) #(255, 0, 255)
25   ROT = (255, 0, 0)
26   ROT_25 = (255, 0, 25)
27   ROTDUNKEL = (100, 0, 0)
28   ORANGE = (200, 100, 0)
29   ROT_100_100= (255, 100, 100)
30
31   SOLUTIONPATHCOLOR = GRUEN_180 #ROT
32   BGCOLOR = BLACK
33   WALLCOLOR_1 = ROT
34   WALLCOLOR_2 = ROTDUNKEL
35   PLAYERPATHCOLOR = GRUEN_115
36   VALIDMOVECOLOR = GRUENNEON
37   INVALIDMOVECOLOR = ROTDUNKEL #ROT
38   STARTCOLER = GRUENNEON
39   ZIELCOLER = ROT_100_100 #ORANGE#PINK#BLAUNEON
40   GENERATOR_COLOR = GRUEN_75
41   BACKTRACKER_COLOR = GRUEN_180 #ROTDUNKEL#PINK
42   #                                  M E S A G E S
43   # mazespiel._get_args() Messages:
44   AXIS_HELP_MSG = " Die Integer-Achsenwerte x-Achse und y-Achse jeweils durch 1 Leerzeichen trennen."
45   GUI_HELP_MSG = "Direkter Start in die Gui für ein einmaliges Spielen, ohne Konsolenausgabe des " \
46                  "Labyrinths."
47
48   # mazespiel.main() Messages:
49   PARAM_MSG = "\n Parametrisierter Programmstart mit x-Achse = {} und y-Achse = {}."
50   NO_PARAM_MSG = "\n Start des Programms ohne Parameter-Übergabe."
51   OVER_2_PARAM_ERRMSG = "\n ERROR: Zu viele Argumente!\n Es wird je 1 x-Achse und 1 y-Achse benötigt!"
```

```
52  ONLY_1_PARAM_ERRMSG = OVER_2_PARAM_ERRMSG.replace("viele","wenige")
53
54  # Konsole.setXYachsen() Messages:
55  MENU = (
56  """
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77  >?> """)
78
79  WRONG_VALUE_ERRMSG = (
80      "\n ERROR: Falls Option 'Eingabe zusammen' gewählt wurde, dann bitte nur:\n"
81      "                                        [y-Achse][x][x-Achse] eingeben\n"
82      " OHNE Leerzeichen & als Trenner nur ein kleines 'x' wie z.B.:\n"
83      "                                                50x50\n"
84      " Bei EinzelEingabe und für die Achsen bitte NUR GANZE POSITIVE ZAHLEN "
85      "eingeben!"
86      "\n")
87
88  TOO_MANY_VALUE_ERRMSG = "\n\t ERROR! Es wurden mehr als 2 Werte eingegeben!\n"
89
90  # Konsole._isAchsenSizeValid() Messages:
91  AXIS_TOO_BIG_ERRMSG = (f"\n\t ERROR! Bei Ihrer Auflösung von {SCREENSIZE[0]}px, "
92      f"{SCREENSIZE[1]}px können Sie maximal für\n\t y-Achse: ^ und für x-Achse: ^"
93      f" eingeben, sodass das generierte\n\t Labyrinth vollständig dargestellt werden"
94      f" kann!\n")
95
96  AXIS_SMALLER_10_ERRMSG \
97      = (f"\n\t ERROR! Die Achsenwerte sollten mindestens 10 betragen,"
98         f"\n\t da die Wahrscheinlichkeit steigt, dass zufällig das"
99         f"\n\t Start- und Zielfeld an der selben Koordinate y,x liegen!"
100        f"\n\n\t Info:{AXIS_TOO_BIG_ERRMSG[9:]}\n")
101
102 AXIS_SMALLER_0_ERRMSG="\n\t ERROR! Achsenwerte dürfen nicht kleiner gleich 0 sein!\n"
103
104 ISVALID_MSG = ("\n\n Validierung der Achsenwerte erfolgreich beendet!\n\n"
105                " In kürze ercheint ein Labyrinth in der Konsole und direkt im\n"
106                " Anschluss die graphische Ausgabe, die entweder als Fenster oder\n"
107                " Vollbild erfolgt.\n Viel Spaß beim Spielen...\n\n\n")
108
109 # MazeSpiel.run() Messages:
110 AUSWERTUNG_MSG = " Auswertung:\n Das Spiel wurde nach {} Sekunden  {}  beendet.\n" \
111                  " Es wurde {} Mal das Start- und Ziel-Feld neu vergeben.\n"
112
113 SOLUTION_MSG = " Spieler Gesamtschrittanzahl: {}\n " \
114                "\t- valide Schrittanzahl:   {} ({} %)\n" \
115                "\t- invalide Schrittanzahl: {} ({} %)\n"\
116                " Lösungspfadlänge: {}\n" \
117                " Spieler-Effizienz: {} %\n\n"
```



Listing 4: konstanten.py

VIII. Eidesstaatliche Erklärung

Hiermit versichern wir, dass wir die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen sind, wurden als solche kenntlich gemacht. Diese Arbeit hat in gleicher Form noch keiner anderen Prüfbehörde vorgelegen.

Frankfurt am Main, den 28. Juni 2021

Stella Malkogiannidou | Samira Maryan Hersi | Amaal Mohamed