Python Programmentwurf

Schiffe versenken

Marcus Unglert, Tobias Schilling, Daniel Balla, Max Domitrovic

Inhaltsverzeichnis

Programmentwurf – Schiffe versenken

1. Einführung

1.1. Verwendete Bibliotheken

- **Keyboard**: Zum Erkennen von Tasteneingaben, womit das gesamte Spiel gesteuert werden kann.
- **Random**: Zum Generieren von zufälligen Werten, um unter anderem eine zufällige Anordnung der Schiffe zu generieren und um die Angriffe des Computergegners zu simulieren.
- **Os**: Um auf das ausführende Betriebssystem zuzugreifen. Unter anderem kann damit erkannt werden, auf welchem Betriebssystem das Programm im Moment ausgeführt wird.
- **Sys**: Kann auf verschiedene Systemvariablen zugreifen und diese verändern. Wir benötigen es unter anderem dazu, um den Pfad am Anfang des Skripts so zu definieren, dass die anderen Module richtige gefunden und geladen werden können.
- **MagicMock:** Zum Simulieren von Tasteneinschlägen innerhalb von Tests.
- **Patch: E**rmöglicht das Ersetzen von Funktionen oder Objekten durch sogenannte Mock-Objekte, wird verwendet um Funktionen innerhalb der Tests durch einen Mock zu ersetzten, um sie während der Tests zu überwachen.
- **Time:** Damit einige geprintete Nachrichten nicht direkt verschwinden, ohne sie gesehen zu haben.

1.2. Spielregeln

- Es müssen insgesamt 10 Boote platziert werden. Eins mit der Länge 5, zwei mit der Länge 4, drei mit der Länge 3 und Vier mit der Länge 2.
- Beim Platzieren der Boote dürfen sich Boote nicht berühren. (auch nicht Diagonal)
- Boote können horizontal und vertikal platziert werden, aber nicht diagonal oder ums Eck
- Das Spielfeld hat die Größe 10x10
- Wenn beide Spieler ihre Boote platziert haben, kann das Spiel beginnen.

- Durch einen Münzwurf wird entschieden, welcher Spieler beginnen darf.
- Trifft der Spieler ein Boot, darf er nochmal angreifen. Solange bis er Wasser trifft.
- Das Spiel ist gewonnen, wenn alle Boote des Gegners zerstört worden sind.

2. Spielverlauf

Beim Ausführen der main.py öffnet sich das Hauptmenü mit den Optionen: Singleplayer, Multiplayer und Exit. Wählt man Singleplayer oder Multiplayer öffnet sich das nächste Menü mit den punkten new game, load game und exit game. Zum Speichern und Laden wird eine JSON-Datei verwendet, welche die jeweiligen Spielfelder speichert. Wird nun ein Spiel gestartet beginnt die Boots Platzierung, man sieht nur sein eigenes Board und rechts davon die Steuerung und darunter die Anzahl an noch zu platzierenden Booten. Bestätigt man seine Platzierungen beginnt das Spiel, es werden Beide Boards angezeigt und man hat rechts wieder die Steuerungen. Links ist das Board des Gegners welches man angreift und links sieht man sein eigenes Feld, welches vom Gegner angegriffen wird. Im Singleplayer spielt man gegen einen Bot welcher immer Zufällige Positionen angreift. Im Multiplayer würde man dann immer die Boards drehen, wenn der andere Spieler am Zug ist, das passiert per Bestätigung der Enter Taste nach einem Angriff, damit man nicht das andere Board sieht. Ist das Spiel zu Ende gespielt worden wird man wieder in das Menü zurückgeleitet und hat die Option wieder zu spielen.

3. Architekturbeschreibung

Grundlegend wurde die Software so entwickelt, dass sie auf allen gängigen Betriebssystemen (Windows, Linux und MacOS) lauffähig ist, unter Linux und MacOS sind eventuell Adminrechte nötig zum Spielen, aufgrund des Keyboard Moduls. Zusätzlich muss auf dem entsprechenden System Python in der Version 3.11 sowie alle zu Beginn aufgeführten Bibliotheken installiert sein.

Um die Programmentwicklung übersichtlicher zu gestalten, wurde der Sourcecode in mehrere Dateien (Module) unterteilt. Die Dateien und deren Funktionen werden im Folgenden beschrieben.

3.1. board.py

In dieser Datei wird die Klasse Board definiert. Sie ist dafür zuständig, die benötigten Spielfelder zu erstellen und anzuzeigen.

3.2. utilities.py

Hier befinden sich Funktionen, die in unterschiedlichen Modulen benötigt werden. Dazu zählen beispielsweise „clear_terminal“, um den aktuellen Inhalt im Terminal zu löschen oder „create_matrix“ und „reset_matrix“, um die Matrizen, mit denen das Spiel gesteuert werden kann, zu erstellen und zurückzusetzen.

3.3. menu.py

Diese Datei beinhaltet die Klasse Menu. Mit der Initialisierung dieser Klasse wird das Hauptmenü erstellt, worüber der Spieler die verschiedenen Modi (Singleplayer oder Multiplayer) auswählen und starten oder das Spiel beenden kann.

3.4. main.py

Dies ist unsere Einstiegsdatei. Sie lädt lediglich alle benötigten Module, um das Spiel zu starten.

3.5. game.py

Hierin befindet sich die eigentliche Spiellogik.

4. Class GameSetup

4.1. __innit__

Board() kreiert ein neues Spielfeld.

current_pos setzt den cursor auf [1, 1]

ammount verfolgt die Anzahl der platzierten Boote

direction setzt die Ausrichtung eines Bootes mit horizontal als Standard

changed_direction prüft, ob das Boot rotiert, wurde

value_matrix ist die Matrix mit den ganzen verschiedenen Werten anhand welchen bestimmt wird, ob es sich um ein Boot oder Wasser handelt, ob das Boot angeschossen wurde, usw.

move_matrix ist die Matrix, auf welcher man den cursor navigiert und Boote platziert

4.2. update_position

Erhält als Parameter die neue Position als Koordinaten ([row, col]), ein bool ob sich das Boot rotiert hat, die Länge des zu platzierenden Bootes und den Namen des aktuellen Spielers.

Es wird geschaut ob nur die Ausrichtung geändert wurde, diese wird geändert und returned. Ansonsten wird die neue Position des Bootes auf der value_matrix aktualisiert und das Board wird aktualisiert und geprintet.

4.3. add_boat_surrounding_horizontal

Erhält als Parameter die Zeile und Reihe des Startpunktes des platzierten Boots, die Bootlänge und den index der Boots Position.

Hier wird alles um das Horizontal platzierte Boot in der value_matrix mit einer 1 versehen damit man dort kein weiteres Boot platzieren kann den das würde gegen die Regeln verstoßen würden sich zwei Boote berühren.

4.4. add_boat_surrounding_vertical

Erhält als Parameter die Zeile und Reihe des Startpunktes des platzierten Boots, die Bootlänge und den index der Boots Position.

Hier wird alles um das Vertikal platzierte Boot in der value_matrix mit einer 1 versehen damit man dort kein weiteres Boot platzieren kann, denn das würde gegen die Regeln verstoßen würden sich zwei Boote berühren.

4.5. change_value

Erhält als Parameter die Länge des platzierten Bootes, dessen Ausrichtung und den momentanen Spieler.

Es wird hier versucht das Boot zu platzieren es wird geprüft, ob das platzieren möglich ist, indem man schaut ob an der Position innerhalb der value matrix schon eine 1 oder 2 steht

was das platzieren blockieren würde. Ansonsten wird es platziert und die Position innerhalb der value_matrix auf 2 gesetzt. Das Board wir aktualisiert und neu geprintet.

4.6. reset_boat_setup

Erhält als Parameter den aktuellen Spielernamen.

Setzt die value_matrix, move_matrix, Ausrichtung und Bootsanzahl zurück.

Aktualisiert das Board und printet dieses.

4.7. change_direction

Erhält als Parameter die Länge des Bootes und den aktuellen Spielernamen.

Ändert die Ausrichtung des Bootes von Horizontal zu Vertikal oder von Vertikal zu Horizontal, indem es die update_position Funktion aufruft.

4.8. handle_key_event_before_placement

Erhält als Parameter ein Keyboard Event, die Länge des Bootes, das ausgewählte Boot (2er oder 3er....) und den aktuellen Spielernamen.

Fängt Keyboard Events ab bevor alle Boote platziert wurden. Hier wird geprüft um was für ein Keyboard Event es sich handelt, bei Bewegungen(up/down/left/right) wird dir update_position Funktion aufgerufen und die Position aktualisiert. Bei enter wird die change_value Funktion aufgerufen und ein Boot platziert falls möglich. Shift rotiert das Boot, R platziert Boote zufällig auf dem Board. Esc setzt das Board zurück, S speichert das aktuelle Spiel und B...

4.9. handle_key_event_after_placement

Erhält als Parameter ein Keyboard Event und den aktuellen Spielernamen.

Fängt Keyboard Events ab nachdem das Boote platzieren fertig ist. Bei Keyboard Event Esc wird die Platzierung zurückgesetzt, S gibt True, True zurück falls alle Boote platziert sind und mit B geht es zurück zur Bootsplatzierung.

4.10. setup_boats

Startet das Boots setup für den aktuellen Spielernamen.

Es beginnt mit einem frisch zurückgesetzten Board und fängt an Keyboard Events abzufangen. Je nachdem ob die Boote schon platziert wurden oder noch nicht wird die Funktion handle_key_event_before_placement oder handle_key_event_after_placement ausgeführt mit dem abgefangenen Key.

5. Class Game

5.1. __innit__

Board() ruft die Klasse Board auf und speichert diese als board.

current_pos setzt den Cursor auf [1,1]

move_matrix wird mit der Funktion create_matrix() erstellt.

win, miss, is_bot werden auf False gesetzt

Confirm_key wird auf einen leeren String gesetzt.

5.2. handle_keyboard_event

Diese Funktion verarbeitet die Tastatur eingaben durch den Spieler. Bei Eingabe von Pfeiltasten, wird die Funktion move aufgerufen, mit der Richtung der Pfeile. Damit es auf mehreren Betriebssystemen läuft wird z.B. nach “up” und nach “nach_oben” überprüft. Bei eingabe der Enter Taste wird überprüft ob der Spieler noch schießen darf, wenn ja wird die Funktion attack aufgerufen mit der aktuellen Position des Cursors. Wenn der Spieler ein Bot trifft, darf die attack Funktion nochmal ausgeführt werden, solang bis der Spieler kein Boot mehr trift. Wenn der Spieler nicht mehr trifft wird der Cursor von der move_matrix entfernt und dadurch nicht mehr auf dem Board angezeigt. Durch wiederholter Eingabe von der Enter Taste wird die Runde beendet. Mit der Esc Taste kann das Spiel beendet und gespeichert werden indem es zwei True Werte zurück gibt.

5.3. update_board

In dieser Funktion wird erst überprüft ob es sich aktuell um einen Spieler handelt oder um einen Bot. Dann werden die Funktionen color_matrix_position_one, color_matrix_position_two und print_double_board aufgerufen und die Matritzen der Spieler und der Movematrix dafür übergeben. Druch diese Funktion wird dann das aktuallisierte doppelte Spielbrett angezeigt.

5.4. move

Bei dieser Funktion wird erst die übergebenen Variable direction abgefragt. Danach wird überprüft ob ein Schritt in diese Richtung noch in dem Spielfeld liegt um ein out of index error zu verhindern. Falls dies möglich ist wird die move_matrix aktuallisiert und das Spielbrett neu geprinten durch das aufrufen der update_board Funktion

5.5. mark_destroyed_boat

Diese Funktion überprüft am Anfang die Richtung des Boates, sprich ob es horizontal oder vertikal auf dem Spielfeld liegt. Dann wird durch zwei Schleifen nach den Treffern der Boote (in der Matrix durch eine 6 dargestellt) gesucht und werden diese zu einem zerstörten Schiff geändert(in der Matrix durch eine 7 dargestellt.

5.6. check_for_game_ending

Mit dieser Funktion wird durch zwei Schleifen die ganze Matrix des Spielfeldes überprüft auf 2er. Wenn eine 2 gefunden wird, also ein ungetroffenes/unzerstörtes Boot, wird False returned. Falls keins gefunden wird, wird ein True returned.

5.7. mark_destroyed_boat_surrounding

Diese Funktion geht mit Hilfe von zwei Schleifen durch das ganze Board und sucht nach der Zahl 7 (7 = zerstörtes Boot). Wenn er diese findet setzt er um die 7 alles auf 5, solang es in dem index liegt und keine andere 7 ist. Da die Boote sich nicht behrühren dürfen fallen die Felder um ein zerstörtes Boot weg, was das Spiel ein bisschen erleichter.

5.8. check_for_destroyed_boat

Mit dieser Funktion wird überprüft ob ein Boot zerstört worden ist. Dabei wird geschaut ob in der Reihe noch 2 er sind und falls ja wird ein false returned. Wenn keine 2 sich in der Reihe befindet wird die aktuelle Position auf 7 gesetzt. Außerdem werden die Funktionen mark_destroyed_boat, mark_destroyed_boat_surrounding und check_for_game_ending aufgerufen.

5.9. attack

Diese Funktion ist für den Angriff vorhanden. Hier wird auf der aktuellen Position in der Matrix nach dem Wert geschaut. Bei dem Wert 5 oder 6 wird ein False returned, da auf der aktuellen Position schon ein mal angegriffen wurde. Bei einer 0 oder 1 wird der Wert auf 5 gesetzt, die Variable miss auf True gesetzt und die Funktion update_board aufgerufen. Bei dem Wert 2 wird der Wert auf 6 gesetztz und miss auf False. Außerdem werden die Funktionen check_for_destroyed_boat und update_board aufgerufen. Die Variable miss hat die Funktion, dass die attack Funktion so lang aufgerufen werden kann bis das Wasser getroffen wurde.

5.10. bot

Diese Funktion holt sich zwei random Int Werte zwischen 1 und 10 und gibt diese der attack Funktion weiter. Dies wird solang wiederholt bis das Wasser getroffen wurde und somit die miss Variabel True ist.

5.11. save_game

Diese Funktion kann aufgerufen werden um das aktuell laufende Spiel zu speichern. Dabei wird erst die save.json geöffnet und überprüft ob ein alter Speicherstand vorhanden ist. Falls ja wieder diesert. Es werden die zwei Spielernamen, die zwei Spieler Matritzen, der Spielmodus und welcher Spieler als nächstes am Zug ist gespeichert.

5.12. delete_save

Diese Funktion versucht nach ablauf eines Singleplayer oder Multiplayer Spieles dieses aus der save.json zu löschen falls diese Vorhanden ist.

5.13. display_text

Hier wird überprüft ob beim aufrufen der Funktion eine 0, 1, 2 oder 3 mittgegeben wurde. Wenn eine 0 mitgegeben wurde, printet er den Spielernamen, der das Spiel startet. Bei Option 1, wird der Spieler geprintet der als nächstes drann ist, genau so wie bei Option 2 für den anderen Spieler. Bei Option 3 wird im Terminal der Gewinner geprintet. Bei allen drei Funktionen wird gewartet bis Enter gedrückt wird bevor es ein return gibt.

5.14. singleplayer

In dieser Funktion passiert der ganze Spielablauf vom singleplayer. Am Anfang wird durch die random.randit enweder die Zahl 1 oder 2, der Spieler gewählt der anfangen darf. Dies soll ein Coinflip simulieren, da es eine 50% Chance ist. Danach folgt eine Schleife in der abwechselnd die Spieler dran kommen und die Funktion handle_keyboard ausgeführt wird. Diese ist wiederum in einer Schleife die endet sobald die Runde vorbei ist. Dann wird überprüft ob der Spieler das Spiel verlassen will oder das Spiel gewonnen hat. Da es sich hier um Singleplayer handelt werden hier die zwei Boards nicht hin und her geswitched da der Bot einfach so angreifen kann und man nicht warten muss bis er fertig ist mit seiner Runde.

5.15. multiplayer

Wie schon bei 5.13 Singleplayer fängt auch hier die Funktion mit einem Coinflip an, der entscheiden soll welcher Spieler anfängt. Danach folgt auch eine Schleife in der abgefragt wird welcher Spieler dran ist, ob der Spieler das Spiel verlassen will oder ob das Spiel gewonnen ist. Auch hier werden wieder die Funktion handle_keyboard aufgerufen

6. Class Menu

6.1. __innit__

Option_input: Speichert die vom Benutzer ausgewählten Optionen. Wird mit einem leeren String initialisiert.

Name_input: Speichert den vom Nutzer eingegebenen Namen. Wird mit einem leeren String initialisiert.

6.2. Main_menu

Stellt das Hauptmenü dar und lässt den Benutzer zwischen "Singleplayer(1)", "Multiplayer(2)" oder "Exit Game(3)" wählen.

Liest eine Benutzereingabe vom Terminal ein und gibt die Eingabe als Ergebnis zurück. Der Benutzer wird aufgefordert, eine der drei Optionen auszuwählen. Wenn der Benutzer einen ungültigen Wert eingibt, wird die Meldung "Invalid input. Please try again." ausgegeben, und der Benutzer wird erneut aufgefordert, eine Option zu wählen. Dieser Vorgang wird so lange wiederholt, bis der Benutzer eine gültige Eingabe tätigt. Sobald eine gültige Eingabe erkannt wurde, wird das Terminal gelöscht, und die Eingabe des Benutzers wird zurückgegeben.

6.3. game_menu

Stellt das Spielmenü dar, nachdem einer der beiden Spielmodi ausgewählt wurde. Hier kann ein neues Spiel gestartet, ein altes geladen oder zum Hauptmenü zurückgekehrt werden.

Funktioniert ähnlich wie die „main_menu“ Methode. Hier werden lediglich andere Optionen angezeigt. Zum Schluss wird der eingegebene Wert zurückgegeben, sofern dieser Gültig ist.

6.4. enter_player_name

Erhält als Übergabeparameter den betroffenen Spieler (1 oder 2) und gibt den dafür eingegebenen Namen vom Benutzer zurück.

Der Benutzer wird aufgefordert, einen Namen für den angegebenen Spieler einzugeben. Wenn der eingegebene Name weniger als 3 Zeichen hat, wird der Benutzer erneut aufgefordert einen Namen einzugeben. Wenn der Benutzer einen Namen eingibt, der 3 oder mehr Zeichen enthält, wird der eingegebene Name zurückgegeben.

7. Class Board

7.1. __innit__

board1 und board2: Matrizen (2-dimensionale Listen), die die beiden Spielbretter darstellen. Werden mit der Größe 12x13 initialisiert.

display_matrix: Matrix, um ein Spielbrett visuell anzuzeigen. Wird mit Größe 11x11 initialisiert.

random_matrix: Matrix, um zufällig platzierte Boote zu speichern. Wird mit Größe 11x11 initialisiert.

value_matrix: Matrix, um den aktuellen Zustand des Bretts während eines Spiels zu speichern. Wird mit Größe 11x11 initialisiert.

Die Methoden add_column_labels und add_row_labels_and_color werden aufgerufen, um die Spalten- und Zeilenbeschriftungen hinzuzufügen.

7.2. add_column_labels

In dieser Methode werden die Spaltenbeschriftungen für das Spielfeld erstellt. Es wird eine Liste von Labeln für jede Spalte erstellt und diese in die erste Zeile der Spielfeldmatrizen eingefügt.

7.3. add_row_labels_and_colors

Diese Methode fügt die Zeilenbeschriftungen hinzu und färbt die einzelnen Spielfelder hellblau.

Zunächst werden dazu die Zeilenbeschriftungen für beide Spieler in die entsprechende Spalte der Matrizen eingefügt. Danach folgt die Färbung für beide Spielfelder.

7.4. color_matrix_positions_boat_setup

Diese Methode färbt die Positionen der Boote, die während der Aufstellung des Spielfeldes platziert wurden. Dabei werden unterschiedliche Farben verwendet, um die entsprechenden Zustände der Boote anzuzeigen:

- Dunkelgrau: Boot wurde platziert und ist nicht beschädigt.
- Blau: Wasser, hier steht kein Boot.
- Rot: Boot kann dort nicht platziert werden.
- Grün: Boot kann hier platziert werden.

7.5. color_matrix_positions_board_one

Diese Methode färbt die Positionen der Schüsse und Treffer auf dem Spielfeld des ersten Spielers. Es wird zwischen verschiedenen Hintergrundfarben unterschieden, um den Zustand der einzelnen Felder anzuzeigen. Als Übergabeparameter bekommt die Methode eine move_matrix und eine value_matrix und erstellt daraus eine display_matrix. Die Bedeutung der Farben ist wie folgt:

- Hellgrau: Feld das angegriffen wurde, aber ohne Treffer.
- Dunkelblau: Getroffenes Boot.
- Die anderen Farben, die auch schon in der color_matrix_positions_boat_setup Methode aufgelistet wurden

7.6. color_matrix_positions_board_two

Ähnlich zur color_matrix_positions_board_one Methode, allerdings wird hier das Spielfeld für den zweiten Spieler gefärbt.

7.7. print_single_board

Hiermit wird das Board im Terminal abgebildet, das beim Platzieren der Schiffe angezeigt wird. Zu dem Spielfeld an sich werden zusätzlich auch eine Anleitung, die die unterschiedlichen Tastenbefehle beschreibt und eine Liste der verfügbaren Schiffe angezeigt.

7.8. print_double_board

Hiermit werden zwei Spielfelder nebeneinander im Terminal ausgegeben. Zusätzlich werden Anweisungen ausgegeben, die dem Spieler zeigen, mit welchen Tasten er welche Aktion durchführen kann.

8. Utilities

8.1. clear_terminal

Überprüft auf welchem Betriebssystem (Linux/Mac oder Windows) das Programm ausgeführt wird und führt dann den entsprechenden Befehl aus, um das Terminal zu bereinigen.

8.2. clear_matrix

Generiert eine leere 11x11 Matrix und gibt diese zurück.

8.3. is_within_bounds

Überprüft, ob beim Platzieren eines Bootes die äußeren Grenzen der Matrix überschritten werden.

Bekommt als Übergabeparameter eine Länge (length), eine Ausrichtung (horizontal oder vertical) und die Reihe (row) beziehungsweise die Spalte (column). Anschließend wird überprüft, ob die Summe aus Länge und Ausgangsposition (Reihe oder Spalte) größer als 10 ist. Wenn ja, wird „False“ zurückgegeben. Wenn nein, wir „True“ zurückgegeben.

8.4. does_not_overlap

Überprüft, ob sich ein Boot beim Platzieren mit einem anderen Boot überschneidet.

Bekommt als Parameter eine Matrix, eine Position mit Reihe und Spalte, eine Länge und eine Ausrichtung, ähnlich wie in der zuvor erklärten Funktion.

Nun wird über die Anzahl der Felder iteriert, die das Boot in Anspruch nimmt, und prüft dabei für jede Position, ob das Boot auf diesem Feld platziert werden kann, ohne dass es sich mit einem anderen Boot überschneidet.

8.5. does_not_touch_vertical

Überprüft, ob ein Boot beim vertikalen Platzieren direkt an einem anderen Boot anliegt. Zwischen zwei Booten muss mindestens ein freies Spielfeld sein.

Der Funktion werden wieder die Parameter Matrix, Position mit Reihe und Spalte und eine Länge übergeben.
Jetzt wird überprüft, ob das zu platzierende Boot ein anderes Boot berührt. Konkret iteriert die Funktion dazu über alle Zellen, die von dem Schiff berührt oder umgeben werden können, indem sie einen Bereich von „length+2“ Zellen überprüft, beginnend mit der Zelle über dem Schiff und endend mit der Zelle unter dem Schiff.
Als Rückgabe gibt die Funktion entsprechend „False“ oder „True“ zurück, je nachdem ob das Boot an der entsprechenden Stelle platziert werden kann.

8.6. does_not_touch_horizontal

Überprüft, ob ein Boot beim horizontalen Platzieren direkt an einem anderen Boot anliegt. Zwischen zwei Booten muss mindestens ein freies Spielfeld sein.

Die Funktion funktioniert ähnlich wie die „Does_not_touch_vertical“ Funktion, allerdings werden hier andere Ausrichtungen und Werte verglichen.

8.7. can_place_boat

Überprüft, ob ein Boot platziert werden kann.

Die Funktion ruft dafür die zuvor aufgeführten Funktionen („Is_within_bounds“, „Does_not_overlap“, „Does_not_touch_vertical“ und „Does_not_touch_horizontal“) auf und übergibt ihnen die entsprechenden Parameter.
Wenn all diese Funktionen „True“ zurückgeben, gibt diese Funktion „True“ zurück.

8.8. add_surrounding_horizonal

Aktualisiert die umgebenden Felder für ein horizontales Boot in der gegebenen Matrix. Die Funktion nimmt eine Matrix, eine Position mit Reihe und Spalte, eine Länge des Schiffes und einen Index i entgegen.

Für jedes Element innerhalb der Länge des Schiffes wird geprüft, ob die umgebenden Felder des Elements innerhalb des Spielbretts liegen. Wenn ja, wird das Element in der Matrix mit dem Wert 1 aktualisiert, um anzuzeigen, dass es von einem Schiff umgeben ist.

8.9. add_surrounding_vertical

Aktualisiert die umgebenden Felder für ein vertikales Boot in der gegebenen Matrix. Die Funktion nimmt eine Matrix, eine Position mit Reihe und Spalte, eine Länge des Schiffes und einen Index i entgegen.

Ähnlich wie in der „add_surrounding_horizonal“ Funktion wird überprüft, ob die umgebenden Felder des Elements innerhalb des Spielbretts liegen. Wenn ja, wird auch hier das Element in der Matrix mit dem Wert 1 aktualisiert.

8.10. random_boat_setup

Diese Funktion generiert eine zufällige Positionierung von Schiffen auf einem Spielfeld.

Zu Beginn wird in der Variable random_matrix eine neue leere Matrix gespeichert. Außerdem werden in „lengths“ die unterschiedlichen Schiffslängen und in „ammounts“ die Anzahl von verschiedenen Schiffen als Liste gespeichert.

Die Funktion generate_random_position() wird verwendet, um zufällige Positionen für Schiffe zu erzeugen. Die Funktion wählt eine zufällige Reihe und Spalte sowie eine zufällige Ausrichtung (horizontal oder vertikal).

Die Funktion place_boat(matrix, row, col, length, direction) platziert das Schiff auf dem Spielfeld.

In der While-Schleife wird nun versucht die einzelnen Schiffe zu platzieren. Dazu wird überprüft, ob ein Schiff an der zufällig gewählten Position inklusive der Ausrichtung platziert werden kann. Wenn ja, wird das Schiff platziert. Wenn dies nicht möglich ist, wird die Variable „attempts“ um 1 hochgezählt. Sollte die Variable „attempts“ den Wert 1000 überschreiten, wird die generierte Matrix mit den bisher platzierten Schiffen gelöscht, „attempts“ wird zurück auf 0 gesetzt und die While-Schleife wird von Beginn an neu gestartet, um eine Endlosschleife zu verhindern.

Konnte ein zufällige Anordnung aller Boote erfolgen, wird die Matrix mit den entsprechenden Werten zurückgegeben.

9. Class Main

9.1. __init__

Game() erstellt ein neues Spiel, GameSetup() ein neues Setup und Menu() ein neues Menü.

9.2. handle_main_menu

Funktion kümmert sich um das Spielmenü und kümmert sich auch um die Benutzereingaben. Leitet die Eingaben weiter an handle_game_menu.

9.3. handle_game_menu

Erhält die vom Benutzer eingebene Zahl welche für 1 ein neues Spiel startet, für 2 ein Spiel lädt und mit 3 geht es ein Menü zurück.

9.4. start_new_game

Erhält als Parameter eine 1 oder eine 2 welche dem zu startendem Gamemode korrespondieren. Bei 1 wird ein neues Singleplayer Spiel gestartet und bei 2 ein neues

Multiplayer Spiel. Die jeweiligen informationen zu dem Spiel wie Spielernamen und Matrixen werden in einer Liste angelegt und an start_game übergeben was das Spiel startet.

9.5. load_game

Erhält als Parameter eine Zahl die mit dem jeweiligem Spielmodus korrespondiert.
Hier wird versucht aus der save.json ein Spielstand zu dem jeweiligem Spielmodus zu ziehen falls gelungen, wird dieser an die Methode start_game übergeben. Falls nicht gelungen kommt der jeweilige error, wie no saved game found oder file is empty.

9.6. start_game

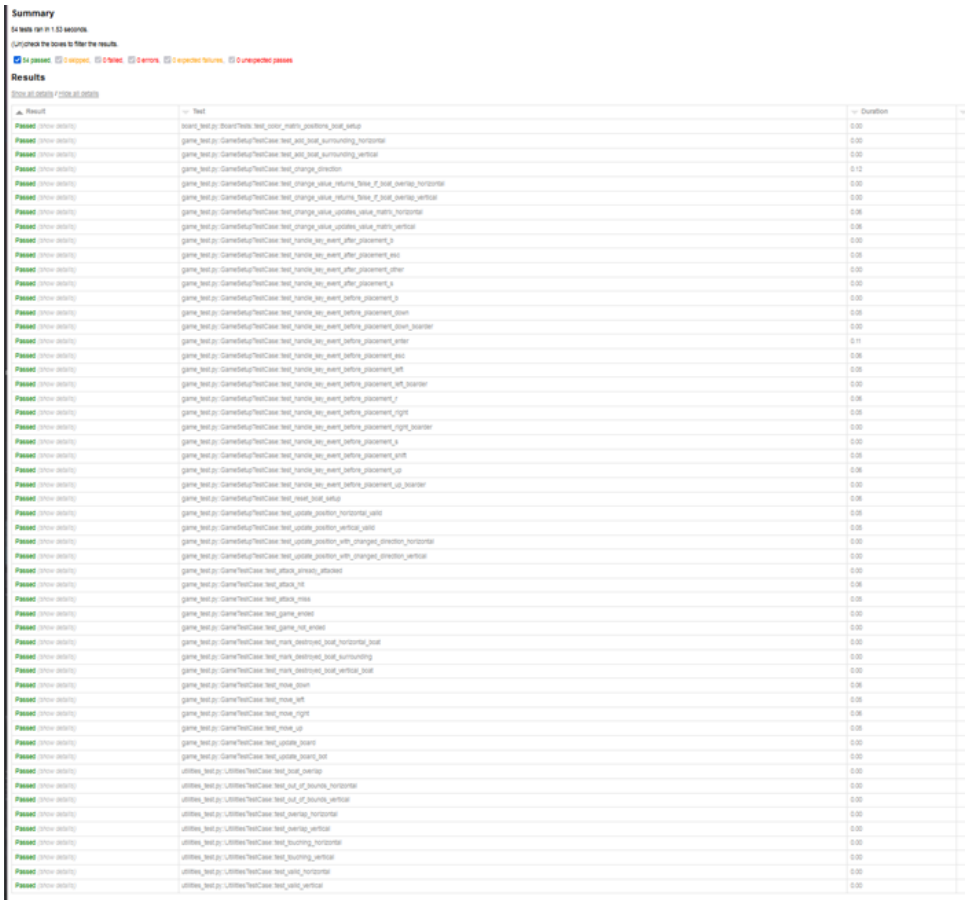
Erhält als Paramter eine Liste von Parametern mit Spielinformation.
Startet je nach Spielmodus, welcher in der Information angeführt ist, entweder den Singleplayer oder den Multiplayer und übergibt diesen wieder die Spielinformationen.

10. Unittest

Die Unittests sind auszuführen mit folgendem Befehl innerhalb des Projektordners:
python -m unittest discover test "*_test.py"

```
............
----------------------------------------------------------------------
Ran 54 tests in 2.433s

OK
```



11. Pylint

```
PS C:\Users\daballa\DHBW\PyCharmProjects\python-projektarbeit-dhbw\test> pylint board_test.py

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

```
PS C:\Users\daballa\DHBW\PyCharmProjects\python-projektarbeit-dhbw\test> pylint utilities_test.py

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

```
PS C:\Users\daballa\DHBW\PyCharmProjects\python-projektarbeit-dhbw\test> pylint game_test.py

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 9.98/10, +0.02)
```

Da es einige komplikationen gab mit den Imports verschiedener Dateien, sowie circular Imports usw., wurden unter pylint die im Screenshot angeführten Meldungen ausgeschalten. Auch ausgeschalten worden ist ‚too-many-public-methods' , da man eben in Tests um alles ausführlich zu testen auch genügend Funktionen benötigt. Das hätte man mit einer weiteren aufteilung in mehrere TestCases lösen können, jedoch ist dies im Kontext nicht nötig.

```
PS C:\Users\daballa\DHBW\PyCharmProjects\python-projektarbeit-dhbw> pylint main.
py
************* Module python-projektarbeit-dhbw.main
main.py:12:0: C0413: Import "from src.game import GameSetup, Game" should be pla
ced at the top of the module (wrong-import-position)
main.py:13:0: C0413: Import "from src.menu import Menu" should be placed at the
top of the module (wrong-import-position)
main.py:14:0: C0413: Import "from src.utilities import random_boat_setup" should
 be placed at the top of the module (wrong-import-position)

--------------------------------------------------------------------
Your code has been rated at 9.62/10 (previous run: 8.35/10, +1.27)
```

Hier wird die Import Reihenfolge bemängelt, jedoch kommt die Warnung bei egal welcher Reihenfolge und lässt sich durch pylint auch nicht abschalten.

```
PS C:\Users\daballa\DHBW\PyCharmProjects\python-projektarbeit-dhbw\src> pylint game.py

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 9.98/10, +0.02)
```

```
PS C:\Users\daballa\DHBW\PyCharmProjects\python-projektarbeit-dhbw\src> pylint menu.py

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 8.75/10, +1.25)
```

```
PS C:\Users\daballa\DHBW\PyCharmProjects\python-projektarbeit-dhbw\src> pylint board.py

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

```
PS C:\Users\daballa\DHBW\PyCharmProjects\python-projektarbeit-dhbw\src> pylint utilities.py

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

12. Coverage

Coverage wird mit folgendem Befehl innerhalb des Projektordners ausgeführt:

coverage run -m unittest discover -s test -p '*_test.py'

Module	statements	missing	excluded	coverage
src\board.py	109	1	0	99%
src\game.py	508	217	0	57%
src\utilities.py	141	6	0	96%
test\board_test.py	17	1	0	94%
test\game_test.py	403	1	0	99%
test\utilities_test.py	55	5	0	91%
Total	1233	231	0	81%

Wir erhalten eine coverage von 81%, hierbei wurden alle relevanten Funktionen getestet. Die in game.py nicht getesteten/covered Funktionen sind die welche zusammengebaut sind aus den ganzen getesteten Funktion so wie z.B. singleplayer, da dies ein Startpunkt ist und einzeln schwer zu testen ist, durch die aufforderung eines Keyboard inputs innerhalb der Funktion.

13. Systemtest

Ausführen von main.py:

Startmenü auswahl Singleplayer:

```
Main Menu:
1. Singleplayer
2. Multiplayer
3. Exit Game

Enter your choice: 1
```

Auswahl neues Spiel:

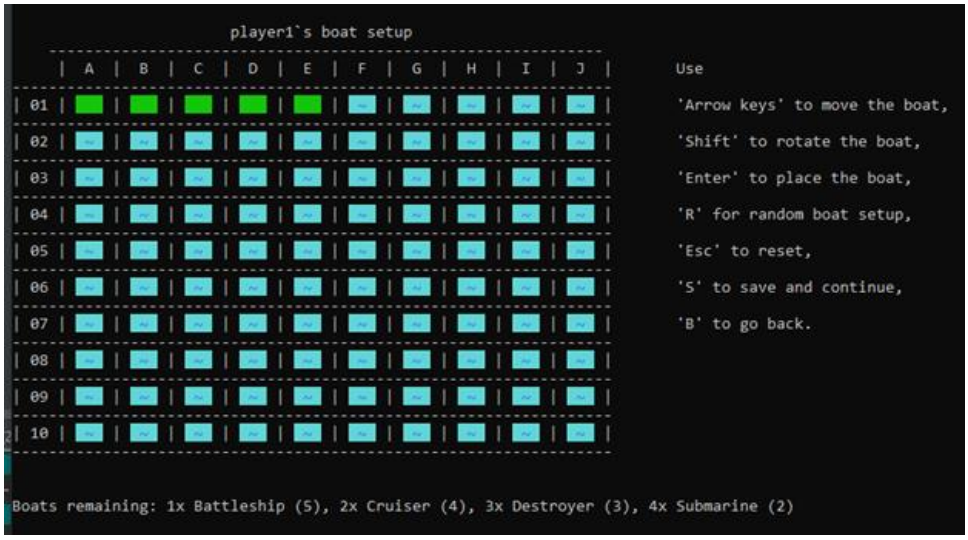
```
Main Menu:
1. New Game
2. Load Game
3. Back to Main Menu

Enter your choice: 1
```

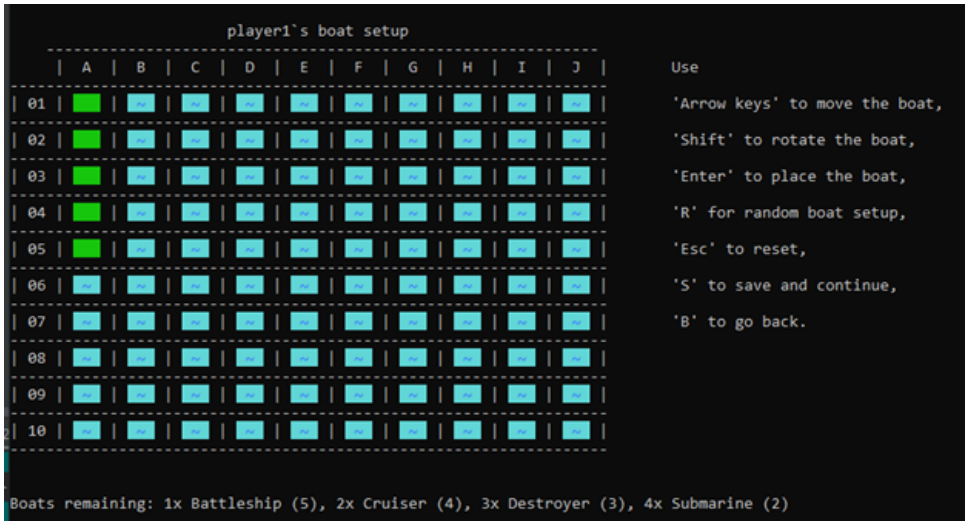
Namenseingabe:

```
Player 1 enter your name: player1
```

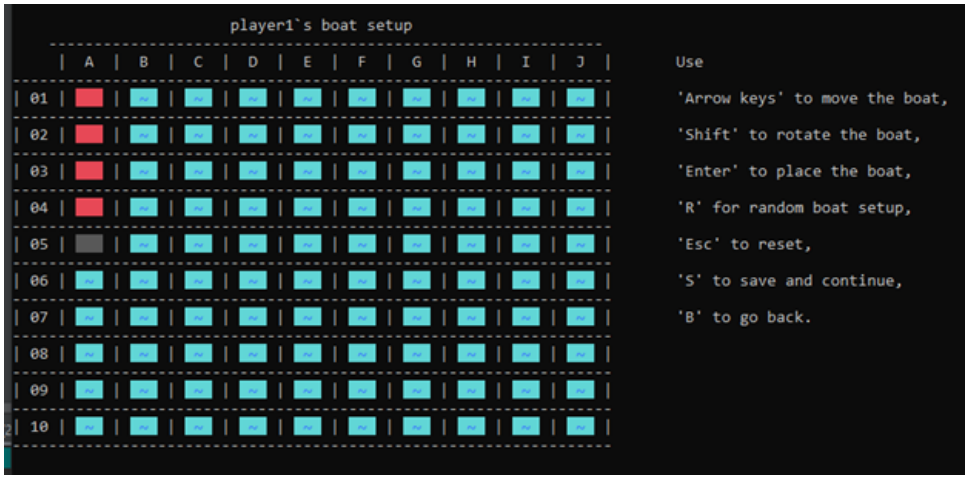
Beginn des Boatsetups:



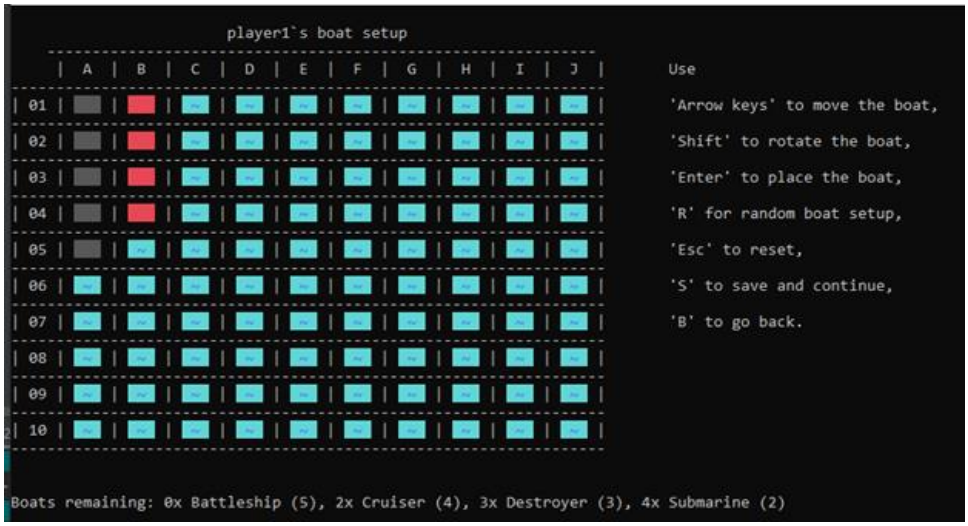
Shift um das Boot zu rotieren:



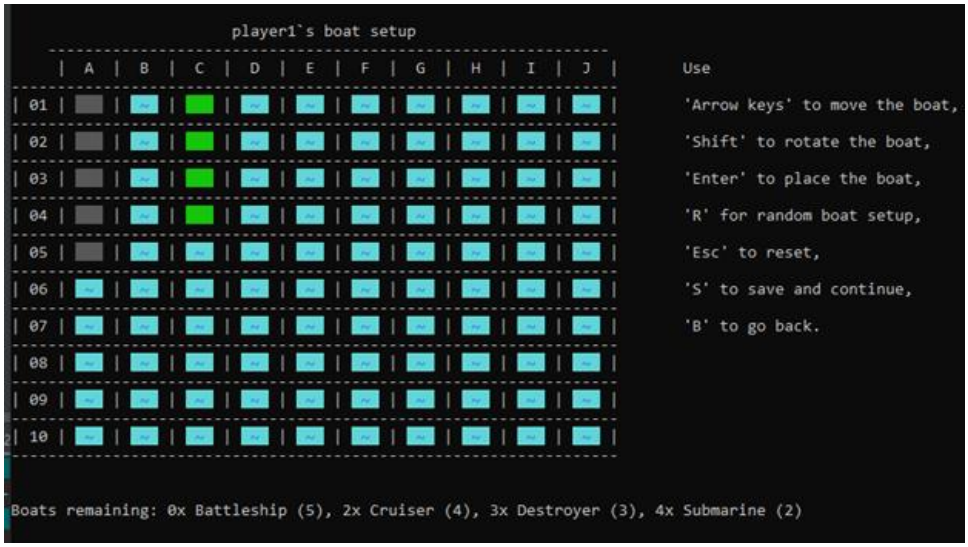
Mit enter wird das schiff plaziert und das neue wird hier in rot angezeigt, da man es nicht auf einem anderem Boot plazieren kann:



Hier sieht man das auch das plazieren genau neben einem anderem Boot auch nicht möglich ist:

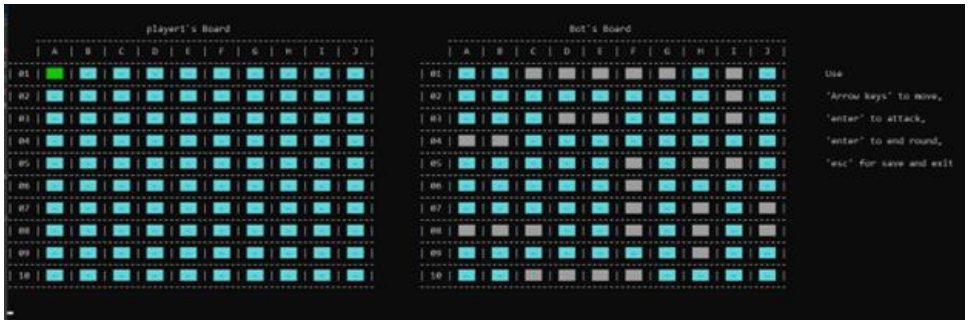


Hier ist es wieder grün, plazieren möglich:

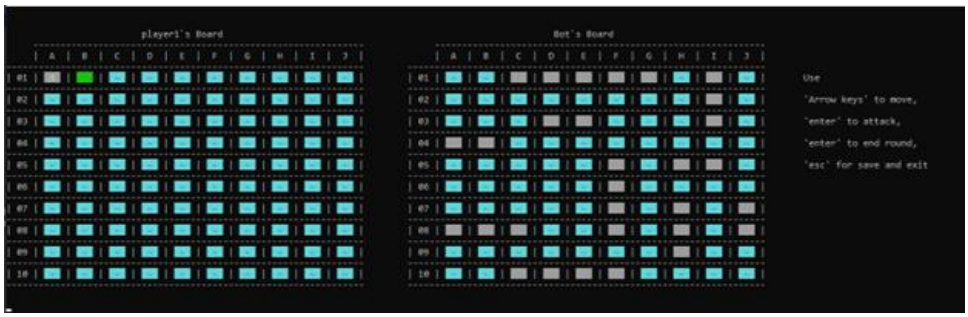


Hier wurde das random boat setup verwendet:

Hier wurde nun das Spiel gestartet:

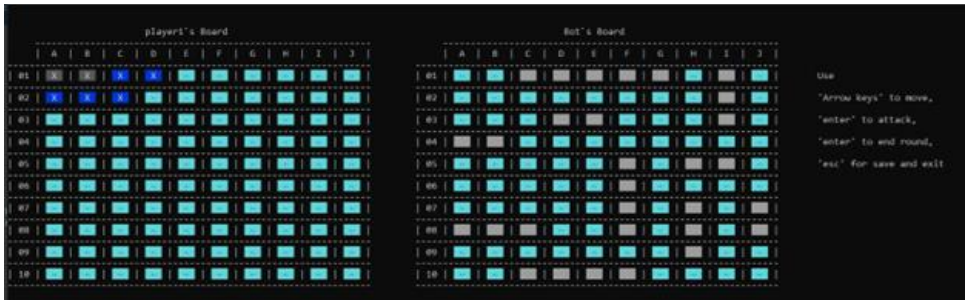


Das graue Feld zeigt das dort ein Boot getroffen wurde:

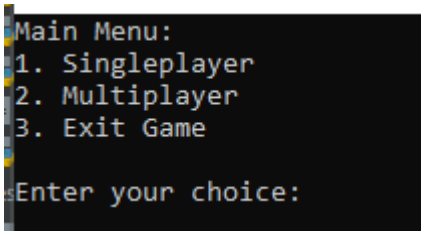


Wurde nun das Boot zerstört werden die Felder blau makiert, da dort kein Boot sein kann:

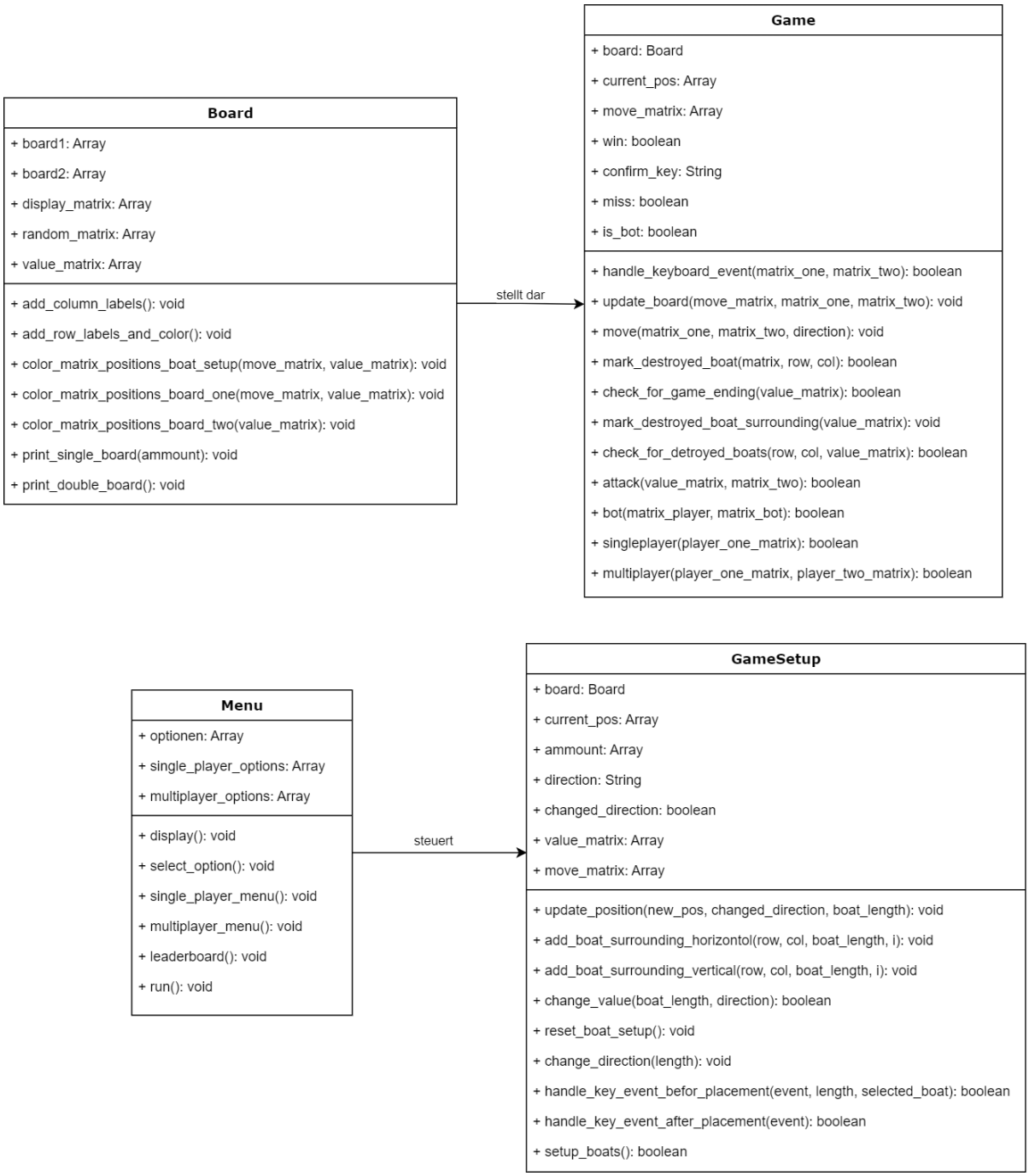
Auch in blau werden Schüsse ins Leere makiert:



Nach dem das Spiel vorbei ist kommt man wieder ins Menü:

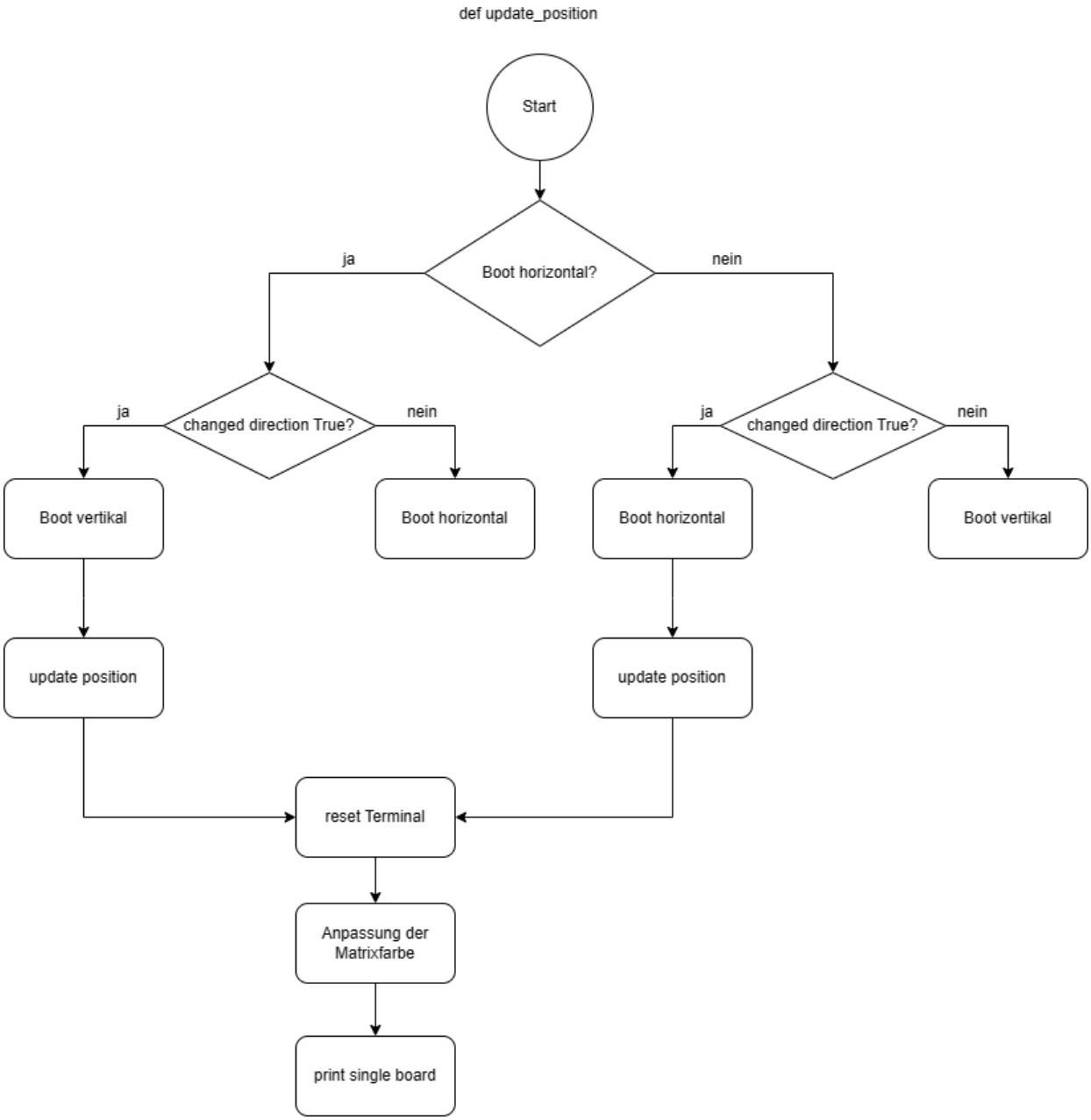


14. Klassendiagramm

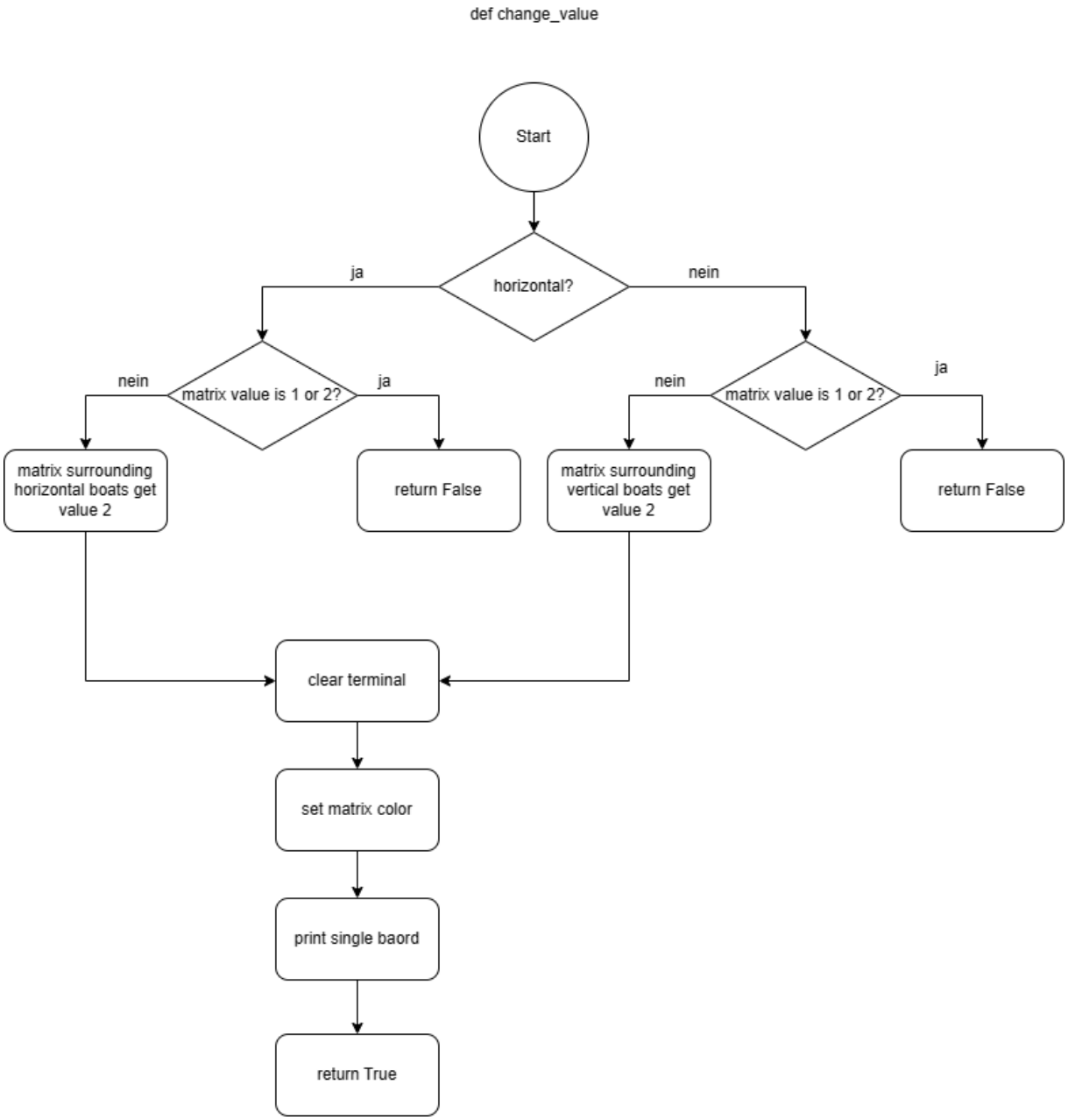


15. Aktivitätendiagramme

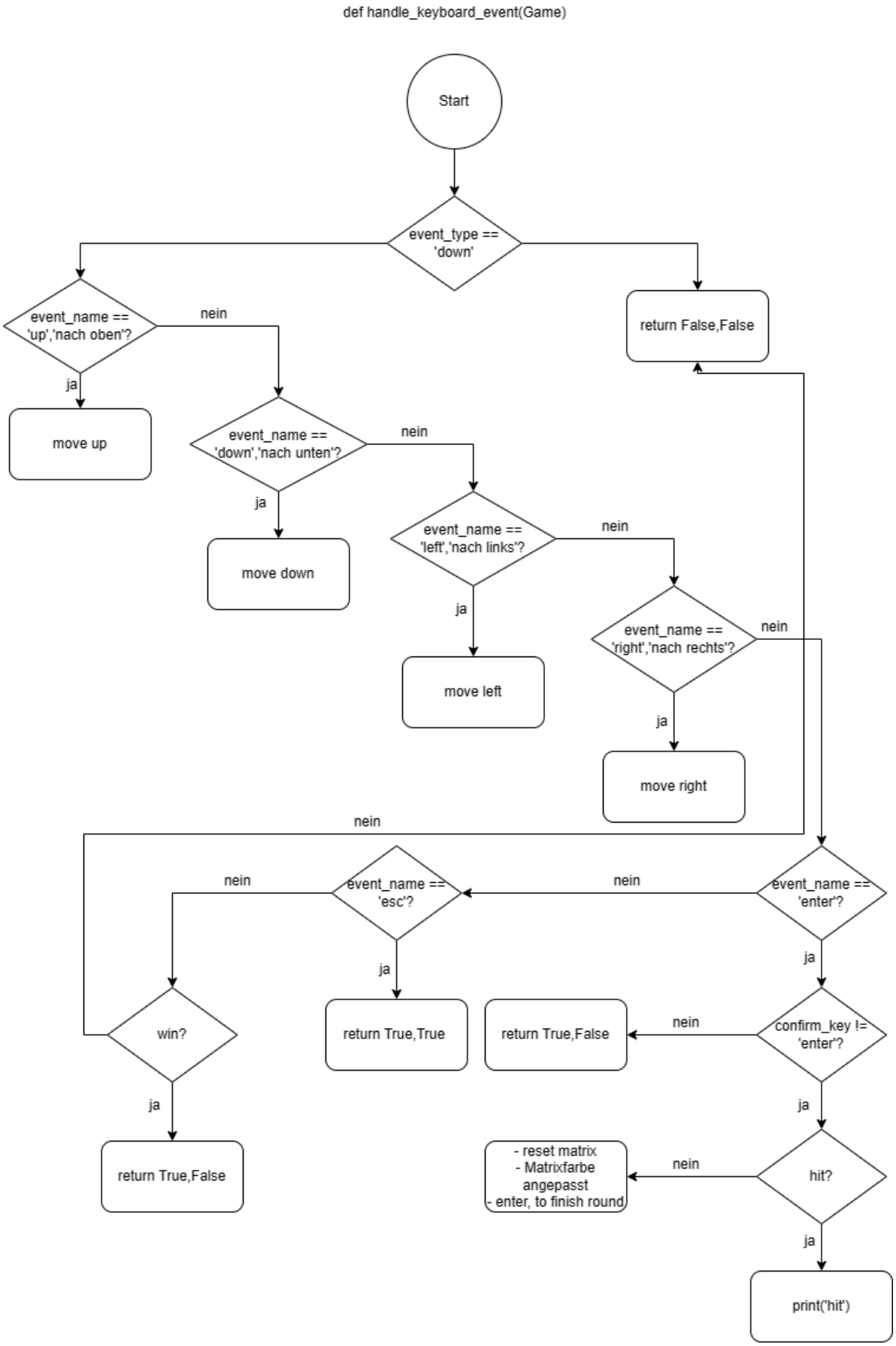
def update_position



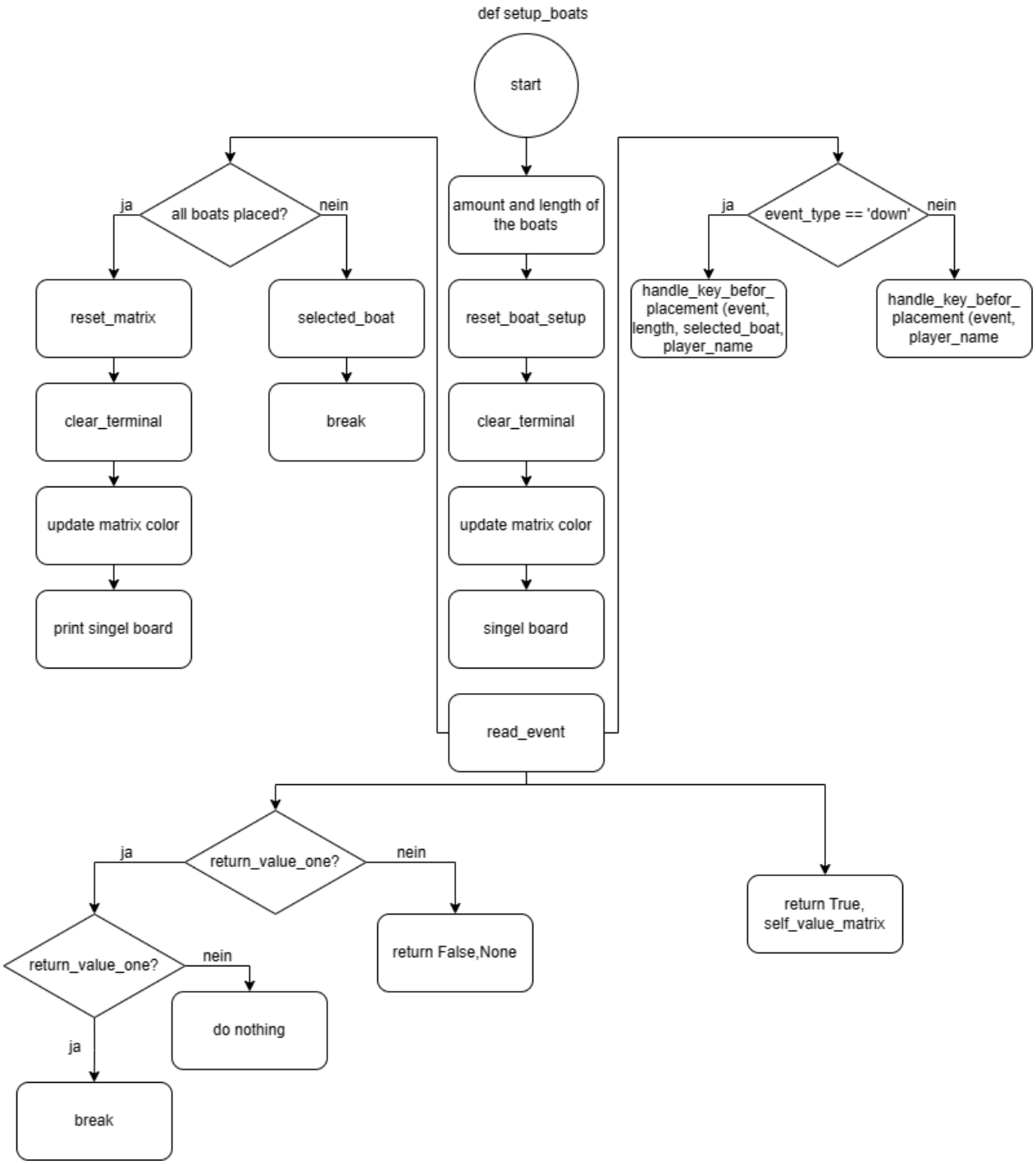
Def change_value



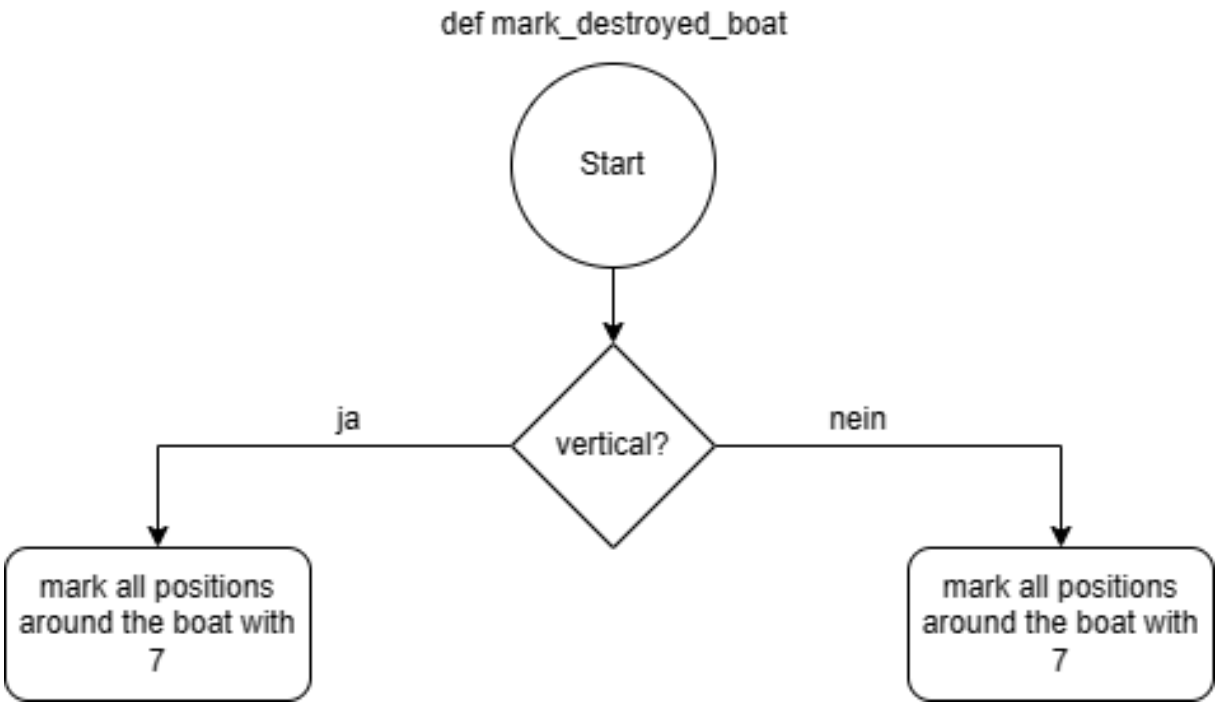
Def handle_keyboard_event



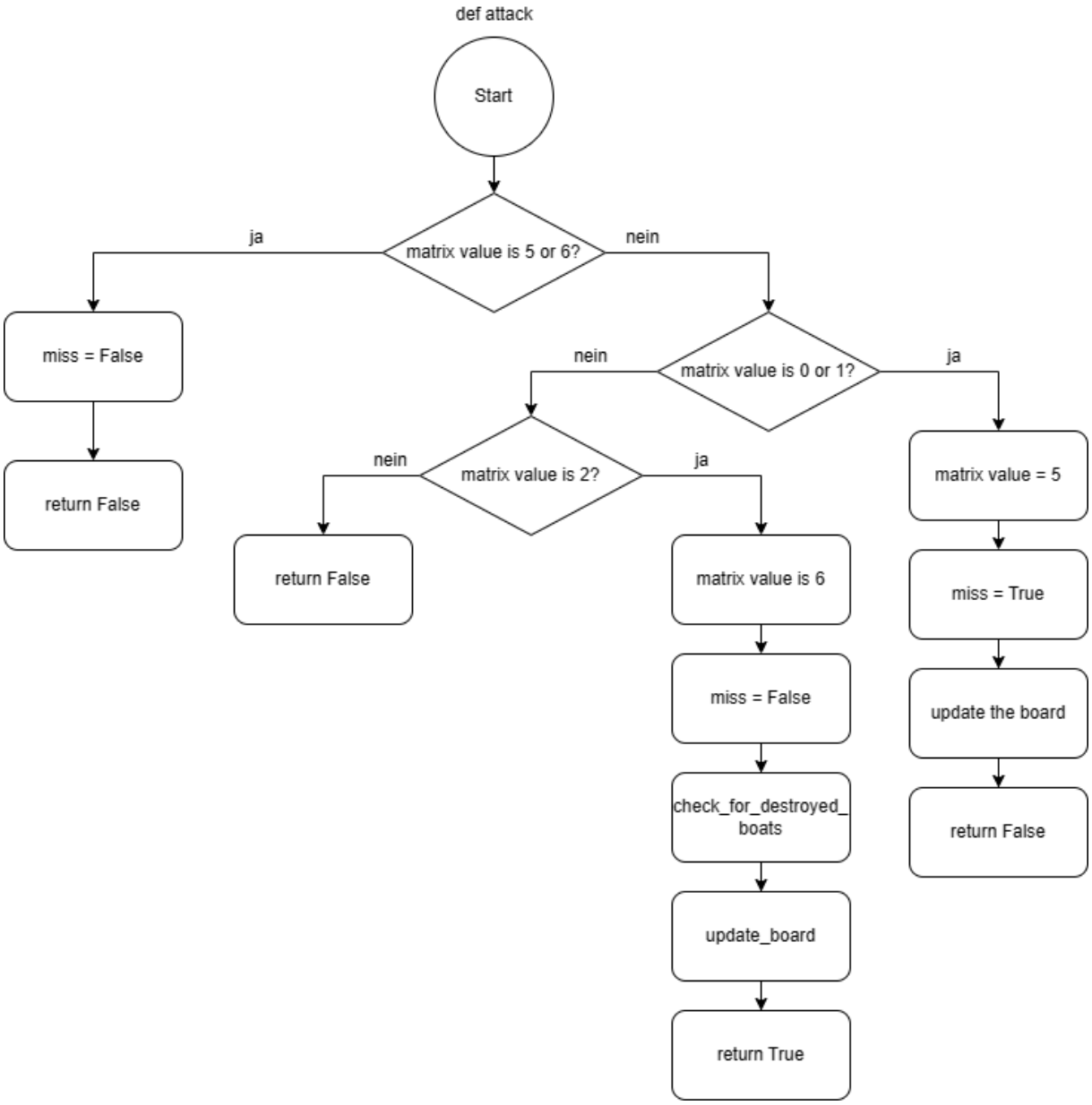
Def setup_boats



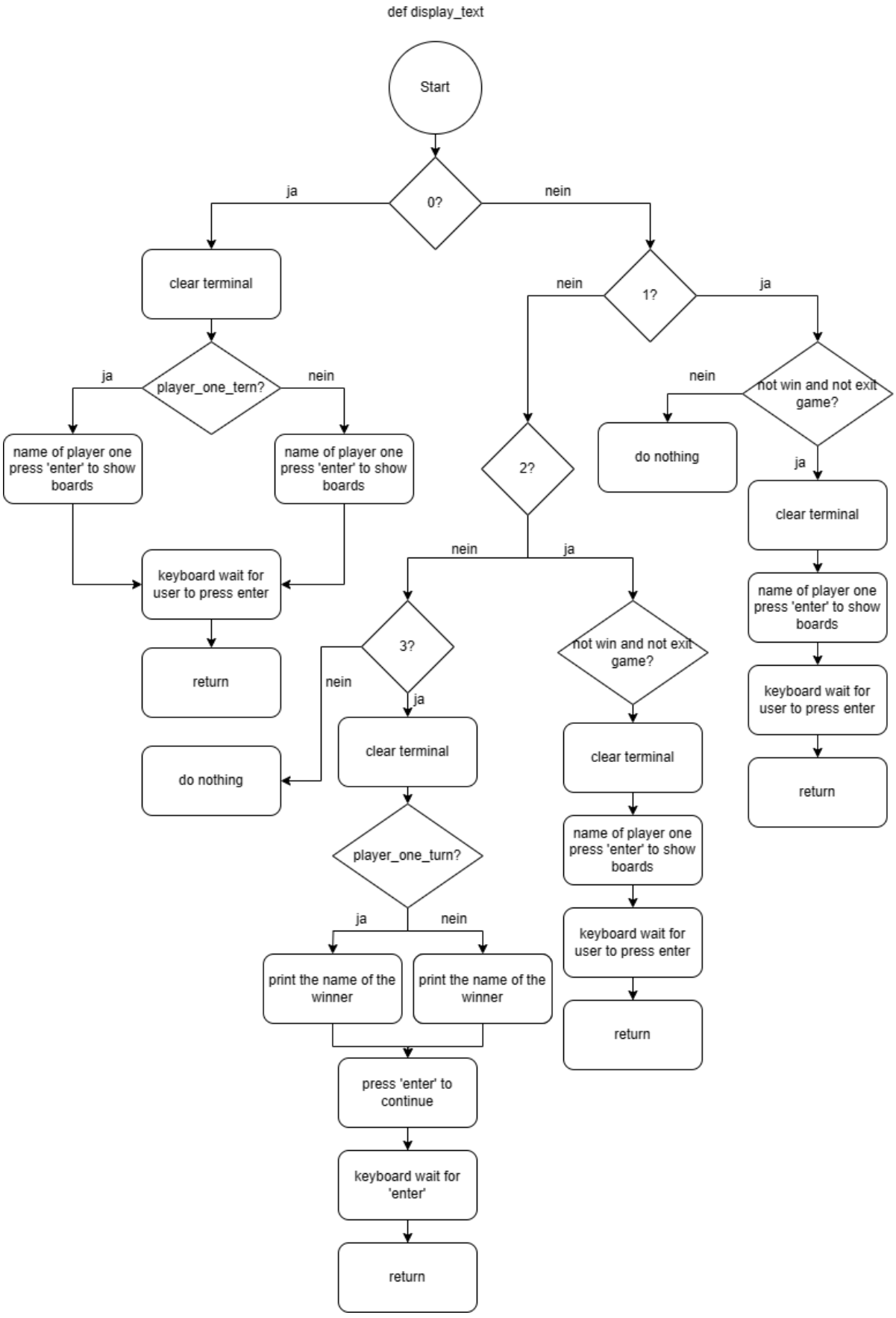
Def mark_destroyed_boats


def mark_destroyed_boat
Start
vertical?
ja
nein
mark all positions around the boat with 7
mark all positions around the boat with 7

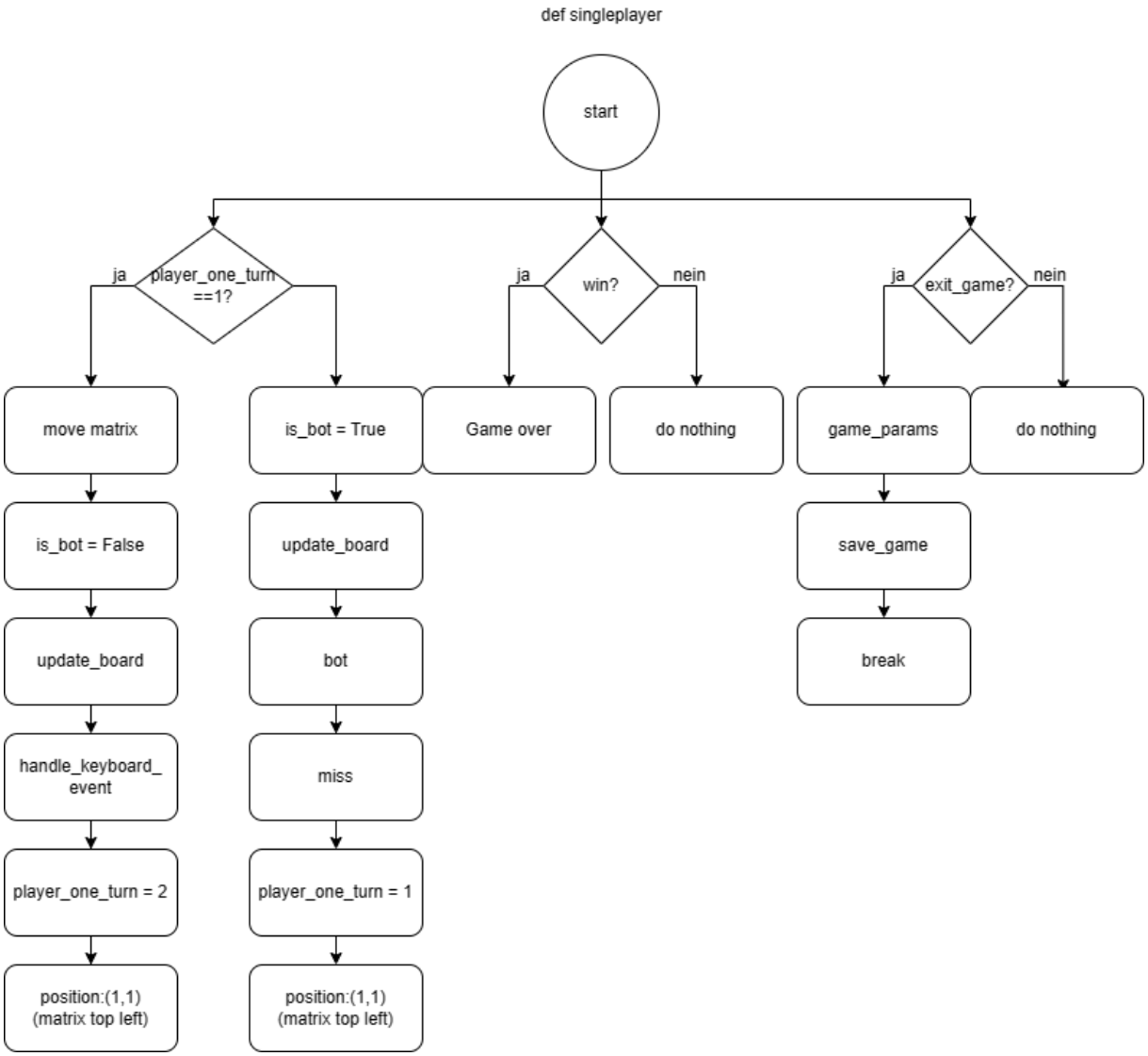
Def attack



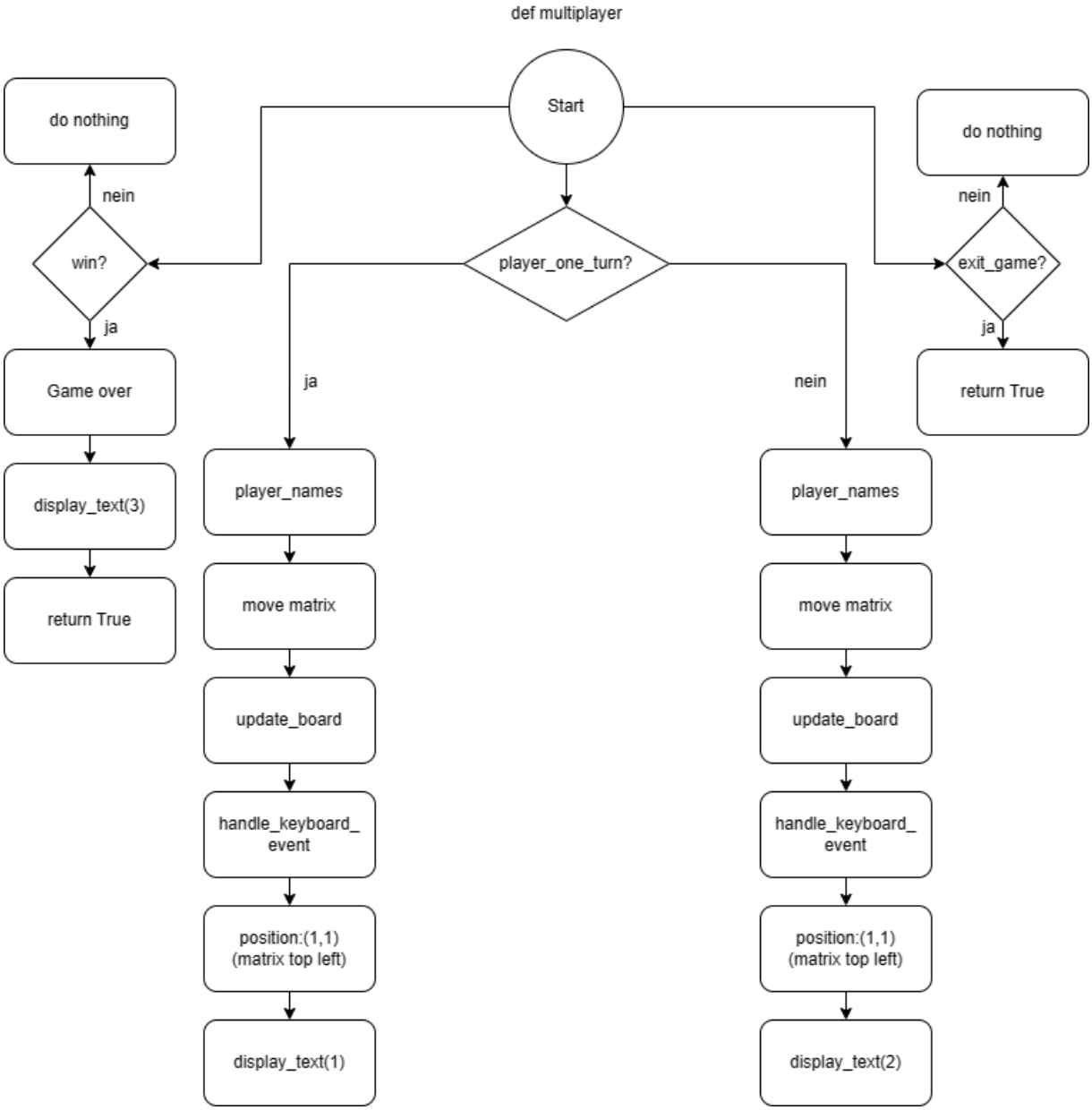
Def display_text



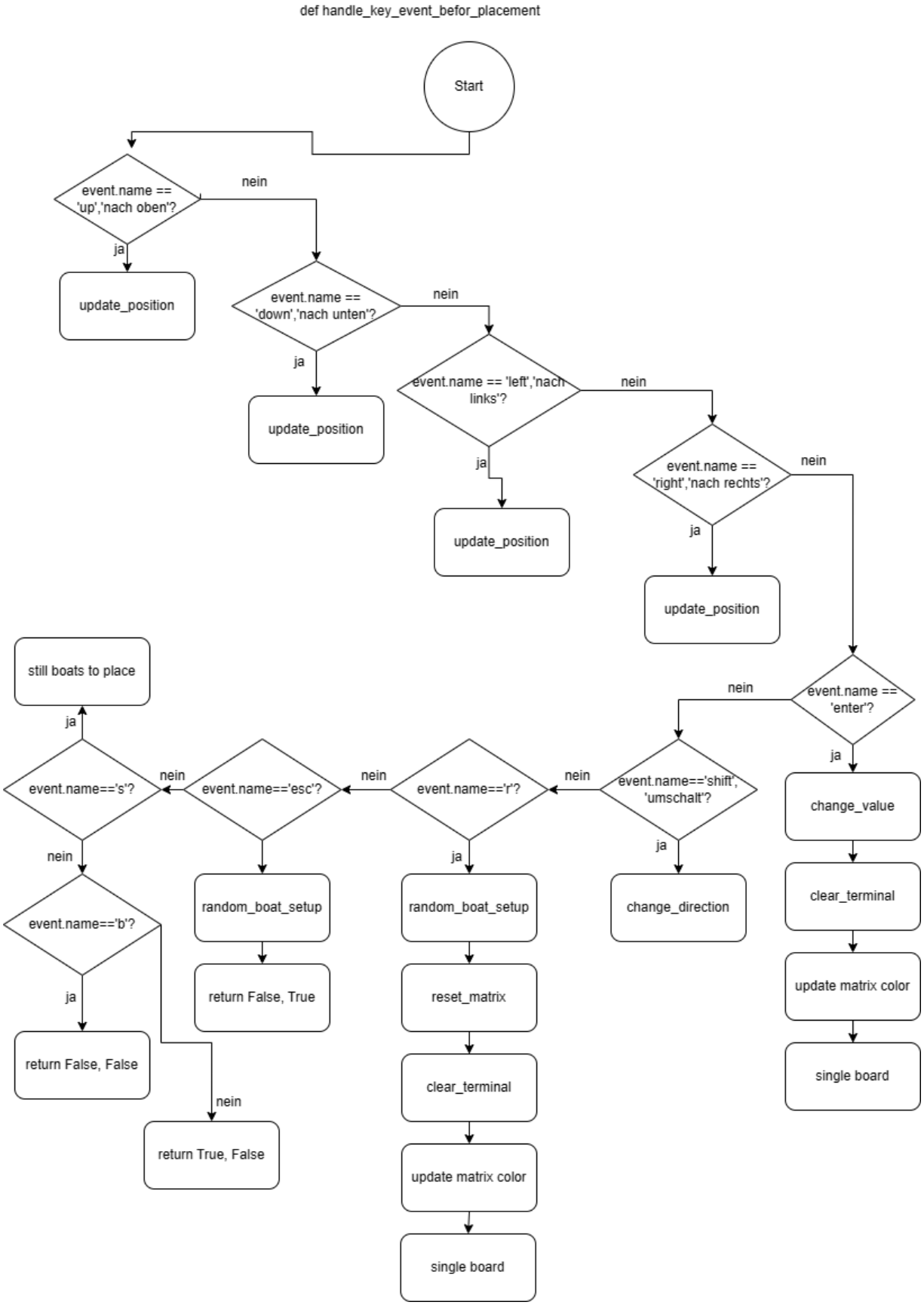
Def singleplayer

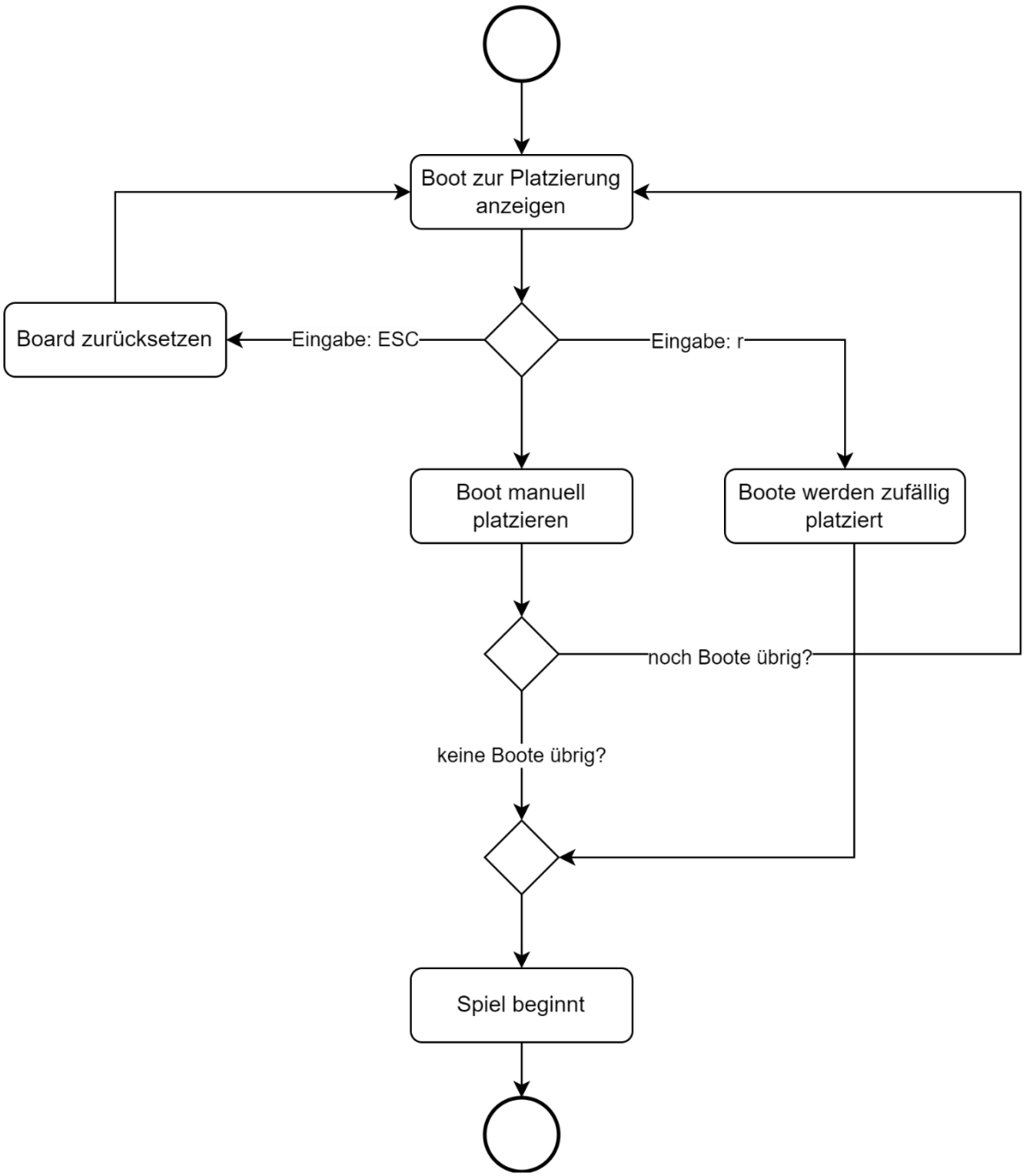


Def multiplayer

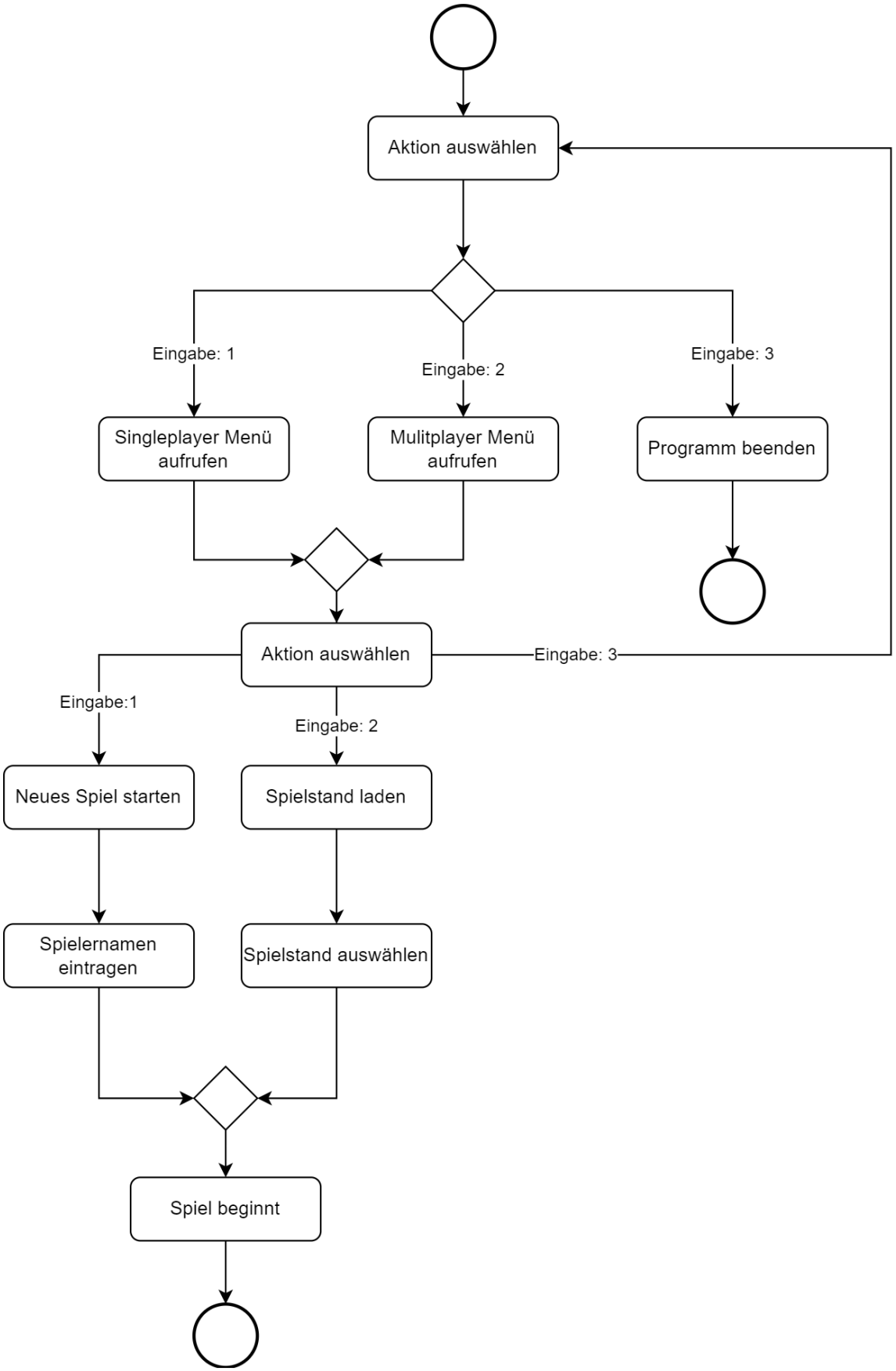


Def handle_key_event_befor_placement


def handle_key_event_befor_placement
Start
event.name == 'up','nach oben'?
nein
ja
update_position
event.name == 'down','nach unten'?
nein
ja
update_position
event.name == 'left','nach links'?
nein
ja
update_position
event.name == 'right','nach rechts'?
nein
ja
update_position
event.name == 'enter'?
nein
ja
change_value
clear_terminal
update matrix color
single board
event.name=='shift', 'umschalt'?
nein
ja
change_direction
event.name=='r'?
nein
ja
random_boat_setup
reset_matrix
clear_terminal
update matrix color
single board
event.name=='esc'?
nein
ja
random_boat_setup
return False, True
event.name=='s'?
nein
ja
still boats to place
event.name=='b'?
nein
ja
return False, False
nein
return True, False



Ablaufdiagramm Board



Ablaufdiagramm Menüführung