

1 Formalitäten

Das Informatik 3 Praktikumsprojekt "Schiffe versenken" von Karl Ruben Kuckelsberg, Pascal Piel und Maurice Mc Laughlin (Matrikelnummern siehe oben).

Das dazugehörige Repository ist unter folgendem Link zu finden: INF3 Schiffe.
(https://github.com/karuku97/INF3_Schiffe)

2 Einführung

Die Aufgabe für dieses Projekt war es, ein Programm zu erstellen, mit dessen Hilfe sich eine bzw. mehrere Strategien für das Spiel "Einbahn Schiffe versenken" bewerten ließ. Dafür sollte ein Server implementiert werden, welcher ein Spielfeld erzeugt, Koordinaten entgegennimmt und entsprechend dem Status des angefragten Feldes eine passende Antwort zurückgibt (siehe 3.1 Protokoll Server). Des Weiteren soll ein Client implementiert werden. Dieser soll eine eigenständig programmierte Strategie, automatisch nach einer Aufforderung und ohne weitere Eingaben eines Benutzers, ausführen bis das Spiel beendet ist. Eine weitere Funktion des Clients soll sein, dass dieser die benötigten Spielzüge zählt, speichert und am Ende ausgibt. Ebenfalls sollen die Strategien auf ihre mittlere Anzahl von Spielzügen verglichen werden.

Das Projekt startete Ende Oktober 2021 und am 04.11.2021 wurden erste Dateien im Repository hochgeladen.

Dabei handelte es sich um die Readme und Dateien, die uns zur Verfügung gestellt wurden. Dazu gehörten: TASK3, Client, Server, Simplesocket und ein Makefile.

Die ersten Strategien wurden am 17.11.21 hochgeladen, einmal Intellistrat und IntellistratDiagonal, sowie Randshoot. Die Funktion Randshoot war zu diesem Zeitpunkt noch als Funktion in der Client-Datei und wurde erst später ausgelagert.

Ende November kamen noch die Strategien Bruteforce und BruteforceDiagonal dazu und die Funktion Randshoot wurde als eigenständige Datei ausgelagert.

Im Dezember wurde noch die Funktion RandshootiS hinzugefügt. Es fanden ebenfalls größere Überarbeitungen des Codes statt, unter anderem die Aufteilung der Strategien in Header- und C++ -Files.

Ende Dezember war die erste Version des Projekts fertig, sodass im Januar 2022 nur kleinere Änderungen des Codes und Bugfixes nötig waren und eine erste statistische Auswertung zur durchschnittlichen Anzahl von Schüssen der einzelnen Strategien möglich war.

2.1 Starten des Programms

Zum Starten des Programms wird erst der Befehl `make all` in die Konsole eingegeben. Danach wird in einer separaten Konsole erst der Server mit dem Befehl `"./Server"` gestartet. In der anderen Konsole wird dann der Client gestartet, dies geschieht durch die Eingabe des Befehls `"./Client 20 3"` die beiden Zahlen sind beispielhaft, die erste Zahl steht für die Anzahl der Spiele die gespielt werden sollen und die zweite Zahl steht stellvertretend für die Methode, die für die Spiele verwendet werden soll.

Strategienummer	Name
0	BRUTEFORCE
1	BRUTEFORCEDIAGONAL
2	RANDSHOOT
3	RANDSHOOTIS
4	INTELLISTRAT
5	INTELLISTRATDIAGONAL

Nachdem Ende jedes Spiel wird die Anzahl der Schüsse in der Konsole ausgegeben und kann zur Auswertung verwendet werden.

3 Strategien und Kommunikation

3.1 Client

Der Client erwartet beim Aufrufen eine zusätzliche Eingabe des Benutzers. Dabei handelt es sich um "Client" und zwei Zahlen. Die erste Zahl gibt an wie oft eine Strategie ausgeführt werden soll, es werden nur Integer akzeptiert. Mit der zweiten Zahl wird die Strategie ausgewählt, es werden nur Integer im Bereich von 0 -6 akzeptiert.

Nach der Eingabe wird die passende Strategie aufgerufen und in einer for-Schleife entsprechend oft der Eingabe ausgeführt.

3.2 Server

Der Server ist eine Kindklasse, die `public` von `TCPserver` geerbt hat. Der Server wurde um die Variablen `world` und `Shootresult` erweitert, welche `private` sind. Eine Funktion des Servers ist `myResponse`, Sie bekommt als Eingabe einen String der vom Client geschickt wird, vergleicht die Koordinaten mit dem internen Spielfeld und gibt einen String als Antwort zurück. Es gibt folgende Antwortmöglichkeiten: "Water", "ShipHit", "ShipDestroyed", "AllShipsDestroyed", "GameOver", abhängig vom internen Spielfeld. Schickt der Client als String "Restart" erstellt der Server ein neues Spielfeld und das Spiel beginnt von neuem.

3.3 Protokoll Server

Zur Kommunikation zwischen Server und Client läuft nach folgendem Protokoll ab.

Server Client Protokoll	Client sends	Server respons
coordinates(shoot)	KORDS + X + {x-coordinate} + Y + {y-coordinate} + #, {x-coordinate} and {y-coordinate} are numbers from 1 until 10, e.g. KORDSX10Y2#	Water, ShipHit, ShipDestroyed, AllShipsDestroyed, GameOver
new Game	RESTART	RESTARTED
end connection	BYEBYE	BYEBYE
error	"no valid command"	ERROR

Wenn der Client eine Strategie ausgeführt, wird bei jedem Schuss ein String mit den zu beschießenden an den Server gesendet und der Server überprüft den Status auf dem internen Spielfeld und sendet eine Antwort in Form eines Strings zurück (siehe Tabelle Zeile 1 Spalte 2). Um ein neues Spiel zu beginnen, muss der Client den String "RESTART" an den Server senden, dieser startet dann ein neues Spiel und sendet den String "RESTARTED" an den Client zurück. Um die Verbindung zu beenden, sendet der Client "BYEBYE", der Server sendet "BYEBYE" zurück und beendet die Verbindung. Sendet der Client einen String, der nicht dem Protokoll entspricht, antwortet der Server mit "ERROR".

3.4 Strategien

Zum beschießen des erzeugten Spielfelds werden 7 unterschiedliche Strategien verwendet.

3.4.1 Bruteforce

Die Strategie BruteForce schießt nacheinander auf alle Zeilen und alle Spalten, bis der Server mitteilt, dass alle Schiffe zerstört sind. Bei jedem Schuss wird ein Schuss Zähler hochgezählt, dieser wird am Ende der Strategie zurückgegeben.

3.4.2 BruteforceDiagonal

Die Strategie schießt nacheinander auf alle Felder im Spiel. Die Reihenfolge der Schüsse ist diagonal angeordnet. Die Strategie wird beendet, wenn der Server mitteilt, dass alle Schiffe zerstört sind. Bei jedem Schuss wird ein Schuss Zähler hochgezählt, dieser wird am Ende der Strategie zurückgegeben.

3.4.3 Randshoot

Bei dieser Strategie wird mittels der `rand()`-Funktion die Koordinaten von 1 bis 10 für beide per Zufall ausgewählt und eine Anfrage mit diesen beliebigen Koordinaten an den Server geschickt. Doppelungen von Anfragen oder mehr als zweimal können auftreten, da nicht gespeichert wird, ob das Feld bereits angefragt wurde oder nicht.

3.4.4 RandshootiS

Diese Strategie ist die Erweiterung zu der `randshoot`-Strategie. Die Generierung der Koordinaten erfolgt wie bei der ersten Strategie. Ergänzend hinzugekommen ist, dass diese Strategie die Anfragen in einem zweidimensionalen Feld abspeichert und nach einer erneuten Generierung prüft, ob diese Koordinaten bereits angefragt wurden. Ist dies der Fall, werden neue Koordinaten generiert. Es erfolgt dann keine neue Anfrage an den Server bei bereits angefragten Koordinaten, die bereits vorher schon angefragt wurden. Fällt das Ergebnis dieser Anfragen negativ aus, so wird eine Anfrage an den Server gestellt. Wird eine Anfrage gestellt, wird der Wert der Koordinaten in der Matrix (2D-Feld) gemäß der booleschen Wahrheitswerte von „false“ auf „true“ geändert.

3.4.5 Intellistrat

Die intelligente Strategie schießt nacheinander (Zeile und Spalte) auf das Spielfeld. Nach jedem Schuss auf ein Feld wird in einer Wahrheitswertmatrix die beschossene Position markiert. Wenn die Strategie einen Treffer erzielt, zerstört Sie zuerst das getroffene Schiff. Dies tut Sie, indem Sie auf die horizontalen und vertikalen Nachbarfelder schießt, bis ihr der Server mitteilt, dass das Schiff zerstört ist. Im Anschluss maskiert Sie die beschossenen und die umliegenden Felder in der Wahrheitswertmatrix, da dort kein weiteres Schiff liegen kann. Bei jedem Schuss wird ein Schuss Zähler hochgezählt, dieser wird am Ende der Strategie zurückgegeben.

3.4.6 IntellistratDiagonal

Die diagonale intelligente Strategie ist auf der einfachen intelligenten Strategie aufgebaut. Der einzige Unterschied besteht darin, dass Sie nur jede zweite diagonale Reihe beschießt. Dies ist möglich, da keine Schiffe existieren, die nur ein Feld groß sind.

3.4.7 Verbrannte Felder

Die Grundidee der Strategie ist, dass sich laut Regeln Regeln des Schiffe Versenkens, in einem Umkreis von einem Feld um ein Schiff kein weiteres befinden darf.

Dies macht sich diese Strategie zu Nutze und markiert bei jedem zerstörten Schiff die anderen Felder in diesem Umkreis schon als Wasser und wird diese dann nicht mehr beschießen. Diese Strategie wird aus zeitlichen Gründen leider nicht mehr fertiggestellt werden können. Jedoch wurden schon bereits Funktionen in der

Klasse "SpielfeldVerwaltung" implementiert, die zu dieser Funktion gehören.

Unter anderem wurde ein zweites eindimensionales Array mit 5 Feldern angelegt,

um dort die Positionen des Schiffes zu speichern welches gerade von der Funktion Nachbar (Erklärung in 3.5 Spielfeldverwaltung) beschossen wird. Die Funktion SchiffPosition wird dafür verwendet, um die Positionen in das Array zu schreiben.

In der Funktion searchShipclass sollte die Anzahl der sich noch im Spiel befindlichen Schiffe gespeichert werden. Dies sollte gemacht werden, um den Abstand der Schüsse zu variieren. Befindet sich noch ein Schiff der Größe 5 Felder im Spiel, so schießt die Strategie in einem Abstand von 5 Feldern. Ist die größte sich noch im Spiel befindliche Schiffsklasse 4 Felder, so wird in einem Abstand von 4 Feldern beschossen.

Diese Abstufung nach unten wird fortgeführt, bis das Spiel beendet wird.

Des Weiteren gibt es noch die beiden Funktionen SchiffPositionToCoordsX und SchiffPositionToCoordsY. Mithilfe dieser Funktionen kann die gespeicherte Position auf dem Array Schiffe zurückgegeben werden.

3.5 Spielfeldverwaltung

Die Spielfeldverwaltung setzt sich aus folgenden Funktionen und einer Klasse zusammen, void restart, enum Feldstatus, string shootPos, Feldstatus shootline, Feldstatus Nachbar und der Klasse SpielfeldVerwaltung. Die Funktion restart wird vom Client benötigt um dem Server mitzuteilen dass das Spiel neugestartet werden soll und shootpos wird verwendet um dem Server mitzuteilen auf welches Feld geschossen werden soll und gibt den Antwortstring des Servers zurück. Unter Feldstatus sind die Enums gespeichert die von der Klasse Spielfeldverwaltung benutzt werden, Erklärung folgt.

Die Funktionen Nachbar und Shootline suchen und zerstören ein Schiff komplett wenn ein Treffer gelandet wurde. Sie funktionieren nach dem Prinzip der Funktion Neighbour welche in der Datei Intellistrat.C verwendet wird und machen sich die Methoden der Klasse zu nutze. Wird ein Schiff getroffen so wird die Funktion Nachbar aufgerufen, sie benötigt als Übergabeparameter die x- und die y- Koordinate, die Anzahl der gezählten Schritte, den TCPClient und eine Referenz auf die aktuelle Instanz der Klasse. Dann wird viermal die Funktion Shootline aufgerufen, Shootline benötigt die eben genannten Übergabeparameter und zusätzlich noch zwei Additionsvariablen. In der Funktion werden dann die Additionsvariablen in einer do-while Schleife auf die x- und y- Variablen aufaddiert, solange der Server ShipHit als Rückgabe liefert.

Dies führt dazu dass in eine Richtung geschossen wird. Wenn Wasser als Rückgabewert kommt wird die Funktion beendet und Nachbar ruft Shootline solange auf bis alle vier

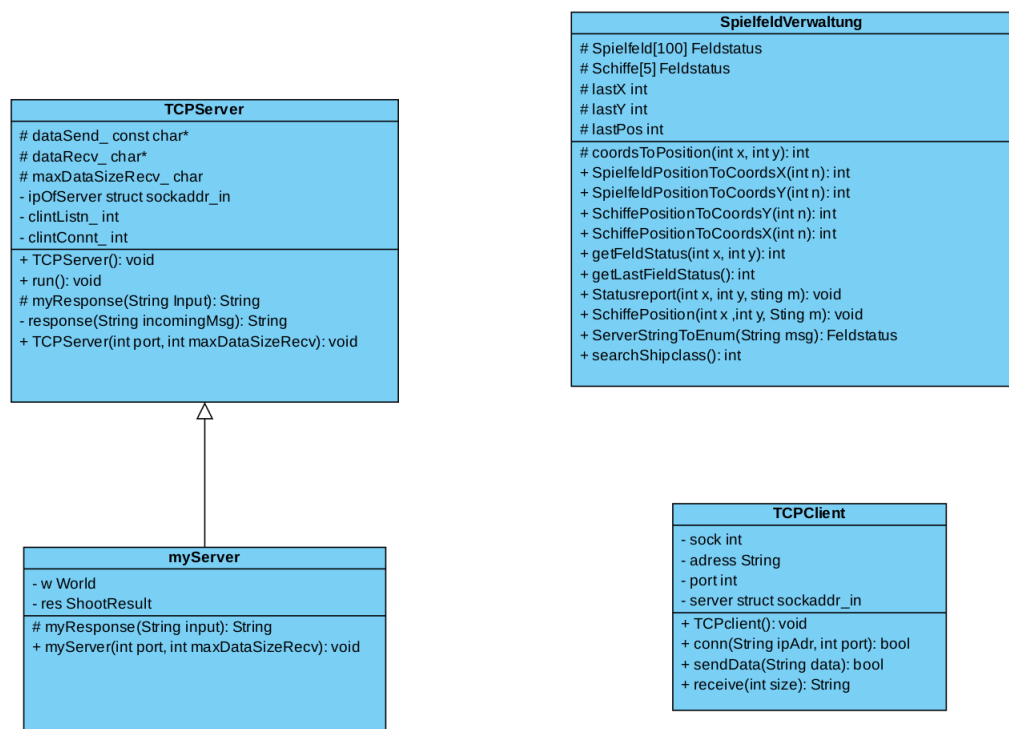
Richtungen ausprobiert wurden bzw. bis das Schiff zerstört wurde, tritt dies schon nach dem ersten aufrufen von Shootline auf wird die Funktion Nachbar ebenfalls beendet.

Die gleichnamige Klasse SpielfeldVerwaltung besitzt ein eindimensionales Array Spielfeld mit 100 Variablen des Typs Enum (die dazugehörigen Enums sind außerhalb der Klasse definiert), die integer Variablen lastX, lastY, lastPos und die Funktionen CoordsToPosition, int SpielfeldPositionToCoordsX, SpielfeldPositionToCoordsY, SchiffePositionToCoordsX, SchiffePositionToCoordsY, getFieldstatus, getLastFieldStatus, Statusreport, SchiffPosition, ServerStringToEnum. Die Funktionen CoordsToPosition, int SpielfeldPositionToCoordsX, SpielfeldPositionToCoordsY, SchiffePositionToCoordsX, SchiffePositionToCoordsY, getFieldstatus, searchShipclass liefern einen Rückgabewert des Typs Integer. Statusreport, SchiffPosition liefern keinen Rückgabewert und ServerstringToEnum liefert einen Enum als Rückgabewert. Das eindimensionale Array Spielfeld, welches in der Klasse protected ist, ist dafür da, um den Status des beschossenen Feldes zu Speichern in dem ein Enum mit folgenden Möglichkeiten NICHT_BESCHOSSEN = 0, WASSER = 1, SCHIFF_GETROFFEN = 2, SCHIFF_ZERSTOERT = 3, GAMEOVER = 4, ERROR = -1 auf die gerade beschossene Stelle schreibt, standardmäßig sind alle Positionen mit NICHT_BESCHOSSEN = 0 beschrieben und werden dann überschrieben. Ermöglicht wird das durch die Funktionen Statusreport und ServerStringToEnum. Wird die Funktion Statusreport aufgerufen, muss ihr die aktuelle Position in Form von einer x- und einer y-Variablen und den String, den der Server beim Beschuss dieses Feldes als Antwort zurückgibt übergeben werden. Nach der Übergabe der Parameter werden die übergebenen x- und y- Koordinaten in lastX und lastY gespeichert, damit in der Klasse immer die aktuelle Position des gerade laufenden Spiels gespeichert ist. Ebenfalls werden die x- und y- Werte von der Funktion CoordsToPosition zusammengerechnet und in der Variablen lastPos gespeichert, diese Variable gibt Auskunft über die aktuelle Position auf dem eindimensionalen Array. Die Funktion CoordsToPosition, welche protected ist, berechnet $((y-1) * 10) + x-1$ und gibt die Lösung zurück. Dann wird die aktuelle Position des Arrays beschrieben, dafür wird auf der Position die mit lastPos festgelegt wurde die Funktion ServerStringToEnum aufgerufen und der String den der Server geschickt hat übergeben. Als Rückgabewert liefert diese Funktion einen des Strings entsprecheden Enum, welcher im Array gespeichert wird. Dieser komplette Vorgang wird bei jedem Schuss ausgeführt, um eine Übersicht über den Spielfortschritt zu haben, um ein doppeltes Beschießen eines Feldes zu verhindern und um zusätzliche Funktionen wie die Nachbar-Funktion zu ermöglichen. Weitere Funktionen der Klasse sind getFieldstatus, welche als Eingabewerte die aktuelle x- und y- Koordinate benötigt und als Rückgabewert den Enum der auf dieser Position im Array steht, liefert. Dies funktioniert ähnlich wie bei der Funktion Statusreport mit der Funktion CoordsToPosition, in diesem Fall wird das Array nicht beschrieben, sondern den Wert der Position zurückgegeben. Die Funktion getLastFieldStatus funktioniert identisch, nur erwartet sie keine Eingabewerte, sondern nutzt als x- und y-Koordinate das, was in den Variablen lastX und lastY gespeichert wurde, der Rückgabewert ist dann wieder der Wert des Arrays an dieser Position. Weitere Funktionen sind SpielfeldPositionToCoordsX und SpielfeldPositionToCoordsY, beide sind identisch sie

erwarten als Eingabewert die aktuelle Position auf dem Array und rechnen dann jeweils die x- oder y-Koordinate aus und geben diese zurück.

Dies ist vor allem beim Debugging sehr hilfreich.

4 UML Diagramme

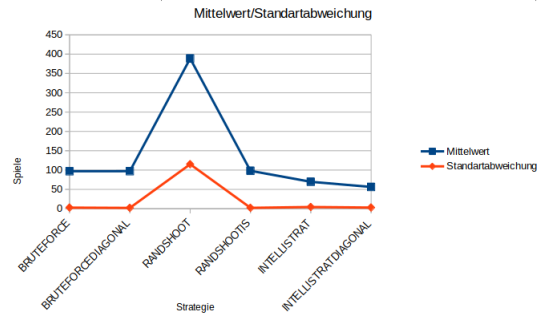
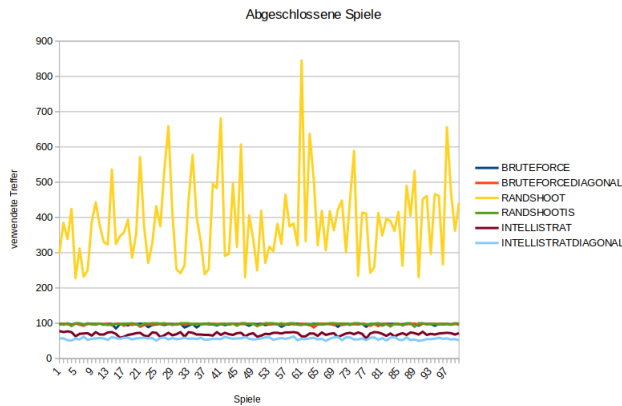


Ergebnisse und Fazit

Durchläufe	BRUTEFORCE	BRUTEFORCEDIAGONAL	RANDSHOOT	RANDSHOOTIS	INTELLISTRAT	INTELLISTRATDIAGONAL
1	99		96	298	99	78
2	98		99	385	96	75
3	99		97	339	100	77
4	96		95	424	92	75
5	99		99	228	100	63
6	99		96	312	100	70
7	96		93	232	99	71
8	98		99	250	100	72
9	99		97	387	98	65
10	98		96	443	99	75
11	99		99	377	99	68
12	96		98	330	97	68
13	99		99	323	95	74
14	98		94	536	97	75
15	85		99	325	98	70
16	98		99	347	98	59
17	97		93	358	99	62
18	94		99	394	99	67
19	99		95	286	100	69
20	97		99	352	98	72
21	99		90	571	93	73
22	98		95	372	100	65
23	89		97	271	99	63
24	95		95	326	100	74
25	97		99	432	100	73
26	98		99	375	99	61
27	95		97	533	100	66
28	98		99	659	98	73
29	98		95	411	98	66
30	97		98	253	96	69
31	99		98	242	100	75
32	88		96	264	100	62
33	93		98	449	100	75
34	98		97	577	97	73
35	88		98	401	99	68
36	97		97	333	98	68
37	99		98	239	98	67
38	97		98	254	100	67
39	97		99	495	98	65
40	94		98	483	96	75
41	98		97	681	99	67
42	95		99	291	98	73
43	97		96	297	99	69
44	99		99	498	99	67
45	98		97	317	93	72
46	98		99	607	100	73
47	98		99	231	100	63
48	93		99	406	98	69
49	99		99	340	99	72
50	97		92	250	93	62
51	99		98	419	96	65
52	95		99	271	100	70
53	97		97	317	100	69
54	98		98	304	100	73
55	98		97	382	97	73
56	90		95	325	100	71
57	96		98	465	98	74
58	97		99	375	100	74
59	99		97	383	100	75
60	99		97	321	96	73
61	97		99	845	95	63
62	97		98	333	98	63
63	95		97	637	98	71
64	99		88	509	96	71
65	97		97	321	100	64
66	97		99	419	99	74
67	99		98	307	98	67
68	98		98	418	100	70
69	98		95	364	100	72
70	90		98	423	99	61
71	99		95	448	99	66
72	99		97	301	99	71
73	97		97	450	95	73
74	99		97	589	100	69
75	97		98	235	100	74
76	99		99	414	98	70
77	90		97	411	97	57
78	98		93	243	97	71
79	98		98	259	99	75
80	99		92	413	100	74
81	97		98	349	93	70
82	99		98	396	98	64
83	98		95	390	91	71
84	99		97	362	99	63
85	97		98	416	99	68
86	97		95	263	94	72
87	99		97	490	100	67
88	98		99	405	100	74
89	99		99	532	99	72
90	93		95	231	100	68
91	99		99	452	100	76
92	98		98	461	97	67
93	98		98	297	98	70
94	93		99	466	98	68
95	98		97	461	97	71
96	97		97	267	99	72
97	98		99	656	98	73
98	97		96	474	96	72
99	99		98	362	100	68
100	99		96	442	99	72

Auswertung

	BRUTEFORCE	BRUTEFORCEDIAGONAL	RANDSHOOT	RANDSHOOTIS	INTELLISTRAT	INTELLISTRATDIAGONAL
Mittelwert	96.91	97.11	388.57	98.02	69.61	56.28
Standardabweichung	2.771624072633	2.06346795468212	115.031061457	2.2089816658361	4.360951730987	2.91231866388278



Bei den sechs Strategien sind teils erhebliche Unterschiede in der Anzahl der durchschnittlich benötigten Schüsse, um ein Spiel zu gewinnen zu erkennen. Die Strategie die die meisten Schüsse benötigt ist Randshoot, dies ist zu erwarten da die Strategie auf zufällig ausgewählte Felder schießt, aber nicht speichert welches Feld bereits beschossen wurde. So ist es unvermeidlich, dass Felder mehrmals beschossen werden. Die Strategie RandshootiS ist eine Erweiterung der Strategie Randshoot, ihre Funktion ist identisch, nur wurde sie erweitert, dass bereits beschossene Felder gespeichert werden, um so ein mehrmaliges Beschießen zu verhindern. Das Ergebnis dieser Erweiterung lässt sich anhand der durchschnittlichen Anzahl der Schüsse und der Standardabweichung beobachten, die Anzahl der benötigten Schüsse und der Wert der Standardabweichung sind signifikant gesunken. Die Strategien Bruteforce und BruteforceDiagonal weisen eine ähnlich hohe Anzahl von durchschnittlichen Schüssen auf, da ihr Grundprinzip das gleiche ist. Die durchschnittliche Anzahl von Schüssen im Bereich von 97 kommt dadurch zu stande, da beide Strategien jedes Feld nacheinander beschießen und es nur 100 Felder maximal gibt. Die Strategien Intellistrat und IntellistratDiagonal sind die beiden effizientesten Strategien. Intellistrat macht sich die Funktion Neighbour zu Nutze, welche bei einem Treffer gezielt nach dem Schiff sucht und es sofort zerstört. IntellistratDiagonal nutzt diese Funktion auch, jedoch durch das Diagonale abgehen der Felder kann jeweils ein Feld horizontal ausgelassen werden und so ein Spiel in noch weniger Schritten gewonnen werden.

Die Random und Bruteforce Strategien weisen eine einfache Code-Struktur, auf benötigen jedoch mehr Schüsse. RandShoot ist ein Extrembeispiel. Während die Intellistrat Strategien komplexere Methoden benötigen, jedoch effizienter und schneller gewinnen.