Informatik 3 Praktikum Protokoll
Karl Ruben Kuckelsberg (1108862)
Pascal Piel (1172238)
Maurice Mc Laughlin (1205943)

Formalitäten

Das Informatik 3 Praktikumsprojekt Schiffeversenken von Karl Ruben Kuckelsberg, Pascal Piel und Maurice Mc Laughlin (Matrikelnummern siehe oben).
Das dazugehörige Repository ist unter folgendem Link zu finden: INF3 Schiffe .

Einführung

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 ebenfals größere Überarbeitungen des Codes statt, unteranderem 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.

Starten des Programms

Zum Starten des Programms wird erst der Befehl "make allïn 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 geschied 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. 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.

Strategien und Kommunikation

Client

Server

Protokoll Server

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

INF3_Schiffe

Project code by Maurice Mc Laughlin , Pascal Piel and Karl Kuckelsberg

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

Strategien

Zum beschießen des erzeugten Spielfelds werden (hier finale Anzahl einfügen) unterschiedliche Strategien verwendet.

Bruteforce

BruteforceDiagonal

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.

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 boolschen Wahrheitswerte von „false“ auf „true“ geändert.

Intellistrat

IntellistratDiagonal

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 Prizip der Funktion Neigbour welche in der Datei Intellistrat.C verwendet wird und machen sich die Methoden der Klasse zur 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 genennte Übergabeparameter und zusätzlich noch zwei Additionsvariablen. In der Funktion werden dann die Additionsvariablen in einer do-while Schleife auf die x- und y- Variablen draufaddiert, solange der Server ShipHit als Rückgabe liefert. Dies führt dazu das 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 aufgerufen von Shootline auf wird die Funktion Nachbar ebenfalls beendet.

Die gleichnahmige Klasse SpielfeldVerwaltung besitz 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 leifern einen Rückgabewert des Typs Integer. Statusreport, SchiffPosition liefern keinen Rückgabewert und Serverstring toEnum 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 folgende

Möglichkeiten NICHT_BESCHOSSEN = 0, WASSER = 1, SCHIFF_GETROFFEN = 2, SCHIFF_ZERSTOERT = 3, GAMEOVER = 4, ERROR = -1 auf die gerade beschossene Stelle schreibt, standartmäßig sind alle Positionen mit NICHT_BESCHOSSEN = 0 beschrieben und werden dann überschrieben. Ermöglicht wird das durch die Funktionen Statusreport und ServerStringToEnum. Wenn die Funktion Statusreport aufgerufen wird 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 protectedist, 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 zurückgegben. 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 SchiffePositionToCoordsX 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 vorallem beim Debugging sehr hilfreich.

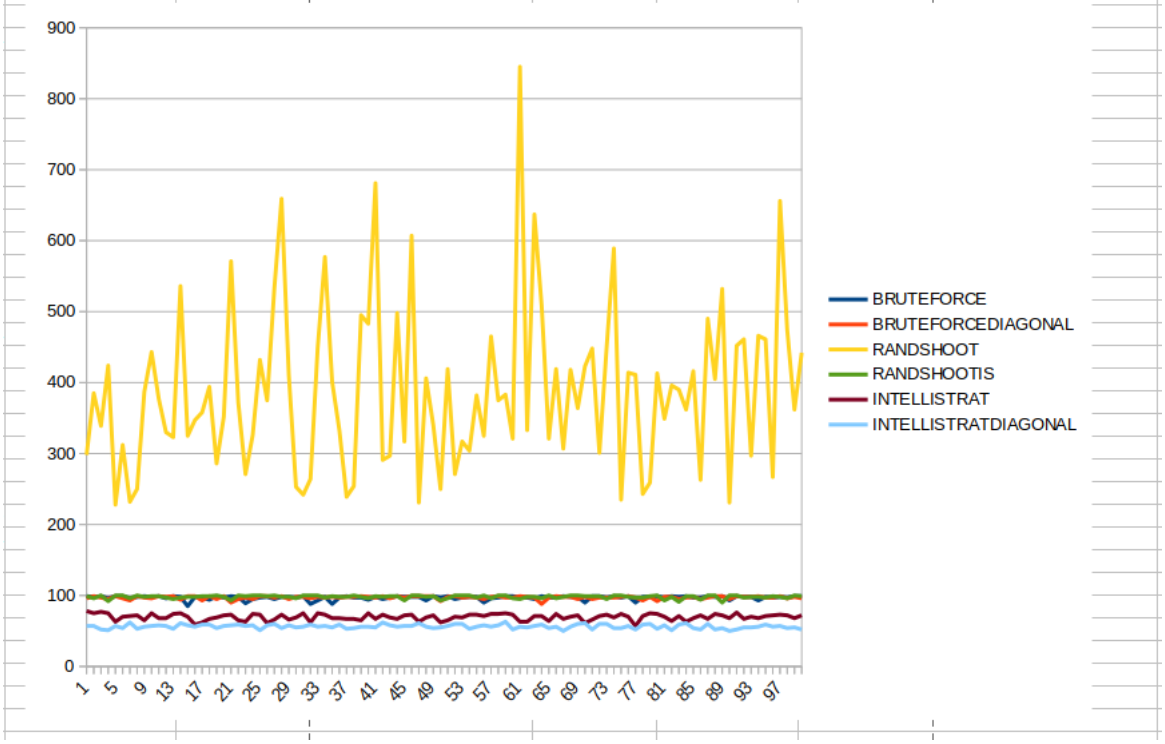
Erklärung der einzelnen Strategien, was unterscheidet die Ideen Protokoll Server Client Kommunikation

UML Diagramme

UML Diagramm einfügen

Ergebnisse und Fatzit

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



Die Ergenisse darstellen und auswerten (erklären wie wir auf die Ergebnisse gekommen sind Standartabweichung etc.)

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 fast 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 Ergebniss dieser Erweiterung lässt sich anhand der durchschnittlichen Anzahl der Schüsse und der Standartabweichung beobachten, die Anzahl der benötigten Schüsse und der Wert der Standartabweichung 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 Startegien 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 Neigbour zu nutze, welche bei einem Treffer geziehlt nach dem Schiff sucht und es sofort zerstört. IntellistratDia-

gonal 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-Strucktur auf benötigen jedoch mehr Schüsse, RandShoot ist ein Extrembeispiel, während die Intellistrat Strategien komplexere Methoden benötigen, jedoch eiffizienter und schneller gewinnen.