P4: Client/Server

Abgabetermin: 26. Mai 2024

Punktzahl: 33 + 3* + 5 (Einhaltung der Coding-Richtlinien)

Inhalt

In diesem Projekt sollen sie das klassische Spiel „Schiffe versenken“ implementieren. In der hierzu zu vervollständigenden Client/Server Applikation sollen zwei Spieler über das Netzwerk gegeneinander spielen können. Der Server agiert dabei als Vermittler zwischen den Spielern und verwaltet den Spielablauf sowie die Kommunikation zwischen den Clients. Jeder Client repräsentiert einen Spieler, der über das Netzwerk mit dem Server verbunden ist. Die Interaktion zwischen den Spielern erfolgt durch das Platzieren von Schiffen auf einem Raster in der Platzier-Phase, sowie das darauffolgende Abfeuern von Schüssen in der Spiel-Phase, um die Position der gegnerischen Schiffe zu erraten. Der Server verarbeitet die Aktionen der Clients, aktualisiert den Spielzustand entsprechend und sendet die relevanten Informationen an die betroffenen Clients zurück.

In der vorgegebenen Archivdatei **p4-vorgaben.zip** finden sie bereits ein Grundgerüst für dieses Spiel. Hierbei fehlen aber noch die Kommunikationsprotokolle, welche sie nun in den folgenden Aufgaben implementieren müssen.

Implementieren Sie zunächst das Serverprotokoll (Aufgabe 1) sowie das Protokoll des Clients (Aufgabe 2). Anschließend implementieren Sie eine einfache Form der Match-Suche auf dem Server (Aufgabe 3). In der letzten Aufgabe (Aufgabe 4) sollen Sie Teile der grafischen Darstellung implementieren, welche das Spielgeschehen veranschaulichen.

Vorgaben

Die Vorgaben in der Datei `p4-vorgaben.zip` beinhalten mehrere Packages mit (zum Teil) vorgegebenen Klassen, die Sie für Ihre Implementierung verwenden sollen und die nachfolgend kurz erläutert werden.

Definition Singleton

Ein `Singleton` ist ein Entwurfsmuster in der Softwareentwicklung, das sicherstellt, dass von einer Klasse genau eine Instanz existiert. Diese Instanz kann statisch über die Klasse selbst aufgerufen werden. I.d.R. besitzen Klassen, die als Singleton angelegt sind, einen privaten Konstruktor und eine Getter-Methode für die einzige Instanz der Klasse). Diese Struktur findet häufig Anwendung in grafischen Applikationen, sowie in Kommunikationsprotokollen. In diesem Projekt wird dieses Entwurfsmuster verwendet und soll auch von Ihnen genutzt werden. Genaueres zu diesem thema erfahren Sie in dem Kurs „Softwaretechnik I“. Klassen, die im folgenden als „Singleton-Klassen“ spezifiziert werden, implementieren dieses Entwurfsmuster.

Package `client`

Dieses Package beinhaltet die **Singleton-Klasse** `Client`, welche die Kommunikation mit dem Server realisiert und die zum Start eines Clients aufgerufen werden muss.

- Die Klasse `Client` besitzt mit den Attributen `input` und `output` einen `ObjectInputStream` und einen `ObjectOutputStream`, welche die Basis für die Kommunikation mit dem Server bilden. Die von der `main`-Methode der Klasse aufgerufende Methode `enterIP` fragt einen Spieler mit Hilfe eines einfachen Pop-up-Fensters nach dem Namen bzw. der IP-Adresse des Servers, startet dann den Client und initiiert die Verbindung mit dem Server. Geben Sie in diesem Fenster daher als Host "localhost" ein (sofern der Server auf der gleichen Maschine läuft). Der Port 3221 ist hier durch eine lokale Variable fest vorgegeben.
- die innere Klasse `client.ServerListener`, empfängt mithilfe der Methode `interact()` alle gesendeten Nachrichten des Servers und wertet diese entsprechend dem Kommunikations-Protokoll aus.

Package `gui`

Dieses Package beinhaltet alle Klassen, die zur Erstellung der Grafischen Oberfläche notwendig sind.

- Die **Singleton-Klasse** `Window` repräsentiert das Fenster der Applikation und ist eine Unterklasse der Klasse `JFrame`. Diese Klasse speichert Informationen über die von der Fenstergröß e abhängige Feldergröße in den Attributen `xSkew` und `ySkew`.

  Diese Klasse enthält zwei für Sie wichtige Methoden:

– **switchToGame()**

Diese Methode ändert die momentan dargestellten Inalte zum Spiel. Hierbei wird die Fenstergröße angepasst und das Spiel gestartet. Schlussendlich wird der Fensterinhalt auf die instanziierte `GameScene` gesetzt.

– **switchToMenu(boolean : waiting)**

Diese Methode ändert den momentan dargestellten Fensterinhalte zum Menü, indem es die Fenstergröße anpasst und der Fensterinhalt auf eine Instanz der Klasse `MenuScene` gesetzt wird. Der Parameter `waiting` bestimmt, ob das dargestellte Menü den Wartebildschirm (`true`) oder das Hauptmenü (`false`) beinhalten soll.

- Die **Singleton-Klasse** `GameScene` repräsentiert die grafische Benutzeroberfläche, die dem Spieler während einer laufenden Spielrunde angezeigt wird. Sie beinhaltet alle interaktiven Komponenten, wie die Spielbereiche der beiden Spieler `p1Panel` und `p2Panel`, mit denen der Spieler während des Spiels interagiert. Zudem speichert das Attribut `phase` die momentane Phase des momentan laufenden Spiels ab. Die statischen Attribute `x1` und `y1` speichern die (Panel-)Koordinaten des momentan ausgewählten Feldes.

- Die Klasse `GameBoard` repräsentiert das zentrale Spielbrett in der `GameScene`. Dieses Spielbrett ist in einen oberen und einen unteren Bereich gegliedert, die durch die zweidimensionalen Arrays `playField` (für das untere Feld) und `enemyField` (für das obere Feld) abgebildet werden. Zusätzlich hält das Spielbrett die aktuellen (Gitter-)Koordinaten der Maus in den Attributen `currentX` und `currentY` fest.

- Die Klasse `Field` ist eine abstrakte Klasse, welche eine einzelne Position (Feld) auf dem Spielbrett darstellt. Diese Position wird durch die Attribute `x` und `y` auf dem Spielbrett beschrieben. Ihr Hauptzweck liegt in der visuellen Darstellung von Treffern und Fehlschüssen bei Angriffen in der Angriffs-Phase. Der Zustand des Feldes wird im Attribut `condition` gespeichert, welches das Enum `Field.Condition` verwendet. Dieses Enum enthält drei Zustände: getroffen (`HIT`), verfehlt (`MISS`) oder noch unentdeckt (`NONE`). Weiterhin hält die Klasse im Attribut `ship` Informationen über den Typ des Schiffs, das sich auf diesem Feld befindet, falls ein Schiff darauf platziert wurde.

  Diese Klasse enthält zudem für sie zwei wichtige Methoden:

– **saveResult(hit : boolean)**

Diese Methode speichert das Resultat eines versuchten Angriffes auf dieser Koordinate ab. Der dazugehörige Parameter `hit` bestimmt, ob der Versuch erfolgreich war (`true`) oder nicht (`false`).

– **getCoords() : int[]**

Diese Methode liefert die relativen Koordinaten dieses Feldes im Verhältnis zum Fensters wieder. Diese (Panel-)Koordinaten werden als Integer-Array mit zwei Einträgen zurückgegeben. Dabei entspricht `int[0]` der x-Koordinate und `int[1]` der y-Koordinate.

- Die Klasse `UpperField` ist eine Unterklasse der Klasse `Field` und repräsentiert ein einzelnes Feld auf dem oberen Teil des Spielbretts. Da das obere Brett ausschließlich für Angriffe auf den Gegner genutzt wird, sendet die Instanz dieser Klasse in der Angriffsphase eine `AttackMessage` an den Server, sobald ausgewählt wurde.

- Die Klasse `LowerField` ist eine Unterklasse der Klasse `Field` und repräsentiert ein einzelnes Feld auf dem unteren Teil des Spielbretts. Da auf diesem Teil des Bretts die eigenen Schiffe platziert werden, wird in der Klasse `LowerField` hauptsächlich die korrekte Darstellung der positionierten Schiffe behandelt.

- Die Klasse `MenuScene` repräsentiert die grafische Benutzeroberfläche, die dem Client angezeigt wird, wenn er sich im Hauptmenü oder im Wartemenü befindet. Die zentralen Komponenten innerhalb der Klasse sind das `waitingPanel` und das `mainPanel`, welche alle wichtigen Elemente für die jeweiligen Fensterinhalte zusammenführen.

Package `messages`

Dieses Package enthält alle Klassen, deren Instanzen über das Netzwerk verschickt werden, um Kommunikation zwischen dem Server und Clients ermöglichen.

- Jede Klasse in diesem Paket ist eine Unterklasse der abstrakten Klasse `Message`, die dank der Implementierung des Interfaces `Serializable` über Streams übertragen werden kann.

- Das Enum `MessageType` bestimmt, um welche Art von Message es sich handelt. Die Art der Message wird direkt im Attribut `type` der Oberklasse `Message` abgespeichert. Jede Unterklasse hat ihren eigenen vordefinierten Typ, der über den Konstruktor der Oberklasse initialisiert wird.

- Die unterschiedlichen Typen von Nachrichten und die Richtung, in der sie versendet werden, lassen sich der untenstehenden Tabelle entnehmen. Bitte beachten Sie, dass

die angegebene Richtung den Weg der Nachricht von ihrem Ursprung bis zu ihrem Ziel beschreibt (zum Beispiel bedeutet „Server -> Client“, dass die Nachricht vom Server an einen Client gesendet wird).

Messages	
AttackMessage StartMessage QuitMessage ResultMessage WinMessage	Server -> Client
ReadyMessage CancelMessage PlacedMessage ShotMessage	Client -> Server

- `AttackMessage` wird gesendet, um den Spieler zu informieren, dass er am Zug ist.
- `StartMessage` wird gesendet, um den Spieler zu informieren, dass ein Spiel gefunden wurde und die Partie starten kann.
- `QuitMessage` wird gesendet, um den Spieler zu informieren, dass sein Gegenspieler die Partie verlassen hat.
- `ResultMessage` wird gesendet, um den Spieler von dem Resultat des zuletzt ausgeführten Schusses zu informieren. Die Klasse enthält die (Gitter-)Koordinaten des anvisierten Feldes in Form der Attribute `x` und `y`, sowie Informationen darüber, ob es sich um einen Treffer handelt (`hit`) und ob der Schuss vom Gegner kam (`byEnemy`).
- `WinMessage` wird gesendet, um einen Spieler über den Gewinner der Partie zu informieren. Das Attribut `opponent` bestimmt hierbei, ob der Gegner der Gewinner ist oder der Spieler selbst.
- `ReadyMessage` wird an den Server gesendet, wenn der Spieler bereit für die Spielersuche ist. In dem Attribut `name` wird zudem der Benutzername des Spielers zwischengespeichert.
- `CancelMessage` wird an den Server gesendet, wenn der Spieler die Spielersuche abbrechen will.
- `PlacedMessage` wird an den Server gesendet, wenn der Spieler alle Schiffe platziert hat. Mit dem Attribut `map` wird dem Server die Platzierung der Schiffe mitgeteilt.
- `ShotMessage` wird an den Server gesendet, wenn der Spieler in der Angriffsphase ein Feld gewählt hat, welches er angreifen möchte. Mit den Attributen `x` und `y` werden die (Gitter-)Koordinaten dieses Feldes weitergegeben.

Package `misc`

In diesem package finden Sie Klassen, die generell wichtig für den Spielablauf sind oder von mehreren Klassen aus unterschiedlichen Packages verwendet werden.

- Das Enum `Phases` legt die verschiedenen Phasen des Spielverlaufs fest. Es umfasst die folgenden Phasen: `FIGHTING` für die Angriffsphase, `PLACEMENT` für die Phase, in der die Schiffe platziert werden, und `NONE` wenn der Spieler nicht an der Reihe ist oder momentan keine Partie stattfindet.
- Die Klasse `RListener` dient lediglich zum überprüfen, ob die „R"-Taste gedrückt wurde. Ist dies der Fall, wird das gewählte Schiff rotiert.
- Das Enum `ShipType` definiert, welche Arten von Schiffen existieren und wie sie aussehen.

  Hierbei sind für sie folgende Methoden wichtig:

  - **getIcon() : Image**

    Diese Methode gibt, in Abhängigkeit des Momentan gewählten Schiffes, eine Instanz der Klasse `Image` wieder, welche das Schiff darstellt. Die zugehörigen Bilder sind dabei im Unterverzeichnis `resources/ships` gespeichert

  - `getSize` () **: int[ ]**

    Diese Methode liefert die Größe des Schiffes in Relation zur Fenstergröße. Im zurückgegebenen Array wird die Breite an der Position `int[0]` und die Höhe an der Position `int[1]` gespeichert.

  - `getRelativeSize` () **: int[ ]**

    Diese Methode gibt die Größe des Schiffes in Form von zwei Werten für die Anzahl von Feldern auf dem Spielbrett (horizontal und vertikal) zurück (also die Gittergröße eines Schiffes). In dem zurückgegebenen Array wird die Breite an der Position `int[0]` und die Höhe an der Position `int[1]` gespeichert.

Package `server`

Dieses Package beinhaltet Klassen, welche sich mit der Erstellung des zentralen Servers, sowie mit der Kommunikation und Verbindung der einzelnen Clients beschäftigt.

- Die Klasse `Server` verwendet eine Instanz der Klasse `ServerSocket`, um einen Port zu öffnen, über den sich die Clients verbinden können. Sobald sich ein Client verbindet, erstellt der Server eine Instanz der Klasse `ClientHandler`.

- Die Klasse `ClientHandler`, eine Unterklasse von `Thread`, ist zuständig für die Verwaltung eines einzelnen Clients, der sich mit dem Server verbunden hat. Die Kommunikation mit den Clients wird über den `ObjectInputStream input` und den `ObjectOutputStream output` realisiert. Die Methode `run()` in dieser Klasse kümmert sich um die Verwaltung aller vom Clients empfangenen Nachrichten und führt entsprechende Aktionen gemäßdem festgelegten Protokoll aus.

- Die innere Klasse `server.MatchCreator` ist eine Unterklasse der Klasse `Thread` und ist für die Verwaltung aller aktuell laufenden Spiele verantwortlich. Die Methode `findWaitingClients()` soll hierbei zwei Clients finden und für sie eine Partie starten.

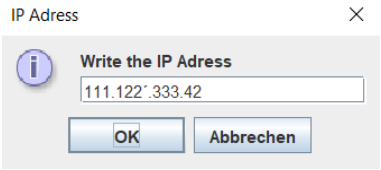
Verzeichnis `resources`

Im Verzeichnis `resources` sind alle für die Darstellung des Spiels benötigten Bilder gespeichert. Dieses Verzeichnis ist in zwei Unterverzeichnisse aufgeteilt: `smiley` und `ships`. Im Unterverzeichnis `smiley` befinden sich alle Smileys, die für die Darstellung der Spieler genutzt werden. In `ships` sind die Bilder für die grafische Darstellung der Schiffe abgelegt. Sowohl die Schiffe als auch die Smileys sind Eigenkreationen und dürfen frei verwendet werden. Zusätzlich enthält das Hauptverzeichnis die Datei `header.png`, die das Logo im Hauptmenü darstellt.

Erklärung des Spiels

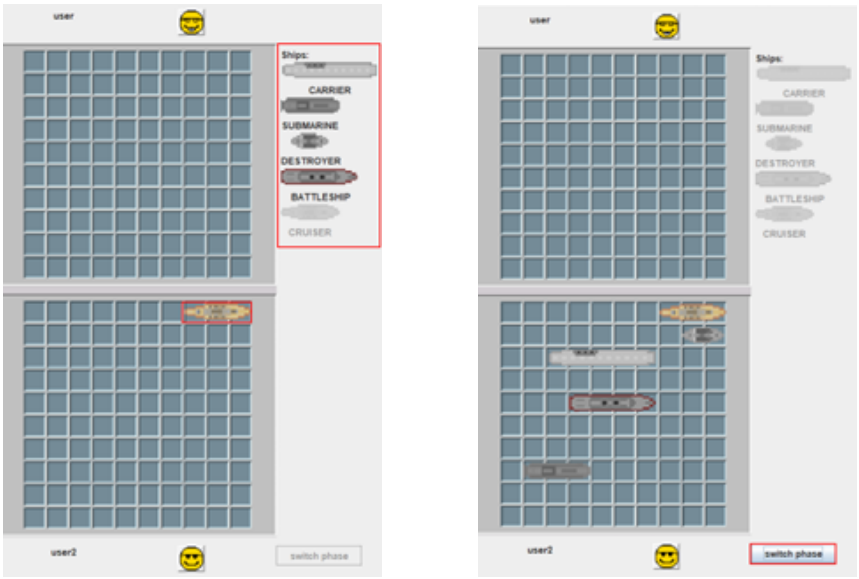
Starten Sie zuerst den Server, indem Sie die Klasse `Server` ausführen. Drücken Sie nun auf den Knopf „OpenConnection“.

Starten Sie nun die Clients. Führen sie hierfür die Klasse `Client` aus. Es öffnet sich automatisch ein Pop-up-Fenster. Geben Sie hier die IP-Adresse des Servers ein (üblicherweise `localhost`), um sich mit ihm zu verbinden.

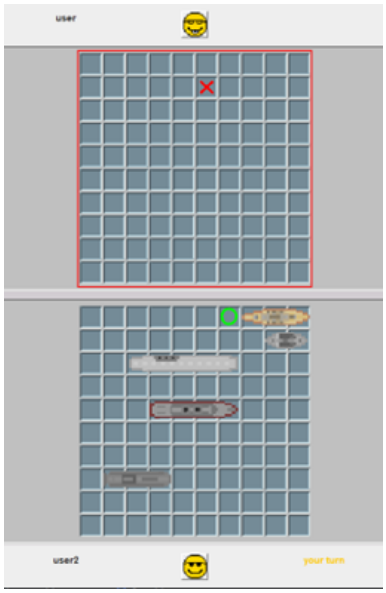


Sie befinden sich nun im Hauptmenü, tragen sie einen Benutzernamen ein und klicken sie auf „Start Game“, um die Suche nach einem Spiel zu beginnen. Beachten Sie, dass Sie das Spiel nur spielen können, wenn ein zweiter Spieler auch nach einer Partie sucht. Führen sie zum Testen alle Schritte nach der Erstellung erneut aus, um einen zweiten Client zu generieren.

Sie befinden sich nun im Spiel. Platzieren Sie zunächst alle die ihnen zu Verfügung gestellten Schiffe. Bewegen Sie hierzu die einzelnen Schiffe auf das untere Spielfeld, indem Sie sie anclicken und dann die Maus über das untere Spielfeld bewegen und mit einem weitern Mausclick fixieren. Wenn Sie während dieses Vorgangs die Taste `R` drücken, wird das Schiff (und ggf alle weiteren Schiffe) um 90 Grad gedreht. Beachten Sie, dass sie alle Schiffe platzieren müssen, bevor sie diese Phase beenden können. Drücken Sie hierfür den Knopf mit der Aufschrift „switch phase“.



Nachdem beide Spieler ihre Schiffsplatzierung bestätigt haben, beginnt die Angriffsphase. Wenn Sie an der Reihe sind, wählen Sie eines der Kästchen im oberen Feld aus, um auf diesem Feld einen Angriff auf den Gegner zu starten. Ob Sie an der Reihe sind, erkennen Sie am Text „Your Turn“, der Ihnen unten links angezeigt wird



Die Spieler wechseln sich so lange ab, bis ein Spieler alle Schiffe des Gegners zerstört hat. Dieser Spieler hat das Spiel gewonnen und die Partie ist zu Ende.

Aufgabenstellung

Aufgabe 1: `ClientHandler` vervollständigen (11.5 P)

Diese Aufgabe beinhaltet das Vervollständigen der `ClientHandler` Klasse aus dem Package `Server`. Der `ClientHandler` ist für Verwaltung eines einzelnen Clients innerhalb des Servers zuständig und regelt den kompletten Kommunikationsfluss mit diesem. Ihre Aufgabe ist es, den Empfang und die Auswertung der Nachrichten, sowie das Senden der entsprechenden Antworten zu implementieren.

Vervollständigen sie hierfür die Methoden `task1()` und `sendMessage` dieser Klasse, welche in der `run`-Methode in einer Schleife aufgerufen wird, die ausgeführt wird, solange der `ClientHandler` aktiv ist

- Zunächst soll mithilfe des Attributs `input` ein Objekt eingelesen werden. Casten sie dieses Objekt auf die Klasse `Message`. Sie können davon ausgehen, dass nur Message-Objekte gesendet werden dürfen.

- Es gibt mehrere Arten von `Message`-Objekten, die anhand ihrer `MessageType` unterschieden werden. Implementieren Sie nun die Handhabung dieser Nachrichtentypen und die zugehörigen Antworten, wie sie in dem folgenden Abschnitt beschrieben werden. Casten Sie dabei das `Message`-Objekt jeweils auf die entsprechende Nachrichten-Klasse und verwenden Sie ggf. die Daten, die diese Nachricht transportiert. Lagern Sie das Senden vom Nachrichten-Objekten an den Client in die Methode `sendMessage()` aus. Vervollständogen Sie die leer vorgegebene Methode `sendMessage()` so, dass die als Parameter übergebene Nachricht an den Client verschickt wird.

  Ready

  – Eine empfangene `ReadyMessage` signalisiert, dass der Spieler die Suche nach einem Gegner starten will. Der passende Gegner hierfür wird durch den `MatchCreator` gefunden. Um den `MatchCreator` zu signalisieren, dass der Spieler bereit für die Spielersuche ist, setzen sie das `isOccupied`-Attribut auf `false`.

  – Speichern Sie zudem den ausgewählten Namen des Spielers im `ClientHandler` in dem dazu vorgesehenen Attribut ab. Der Name ist in der Nachricht gespeichert.

  Cancel

  – Diese Nachricht wird gesendet, wenn der Spieler sich entscheidet, die Spielsuche abzubrechen. Setzen Sie in diesem Fall das `isOccupied`-Attribut wieder auf `true`.

Placed

- Diese Nachricht wird gesendet, sobald der Spieler alle seine Schiffe platziert hat. Innerhalb dieser Message wird die Karte der platzierten Schiffe als 2D-Array mit den Ordinalwerten der Schiffe (aus dem Enum `ShipType`) mitgesendet. Speichern Sie dieses Array im `opponentMap`-Attribut des Gegners, welcher über das `opponent`-Attribut referenziert wird.
- Passen Sie nun den Wert des `placed` Attributs, welches signalisiert, ob ein Spieler all seine Schiffe platziert hat, entsprechend an.
- überprüfen sie nun mit der Methode `hasPlaced()`, ob ihr Gegner auch alle Schiffe platziert hat. Ist dies der Fall, kann die Angriffsphase beginnen. Um zu entscheiden, wer zuerst angreifen darf, wird virtuell eine Münze geworfen. Sertzen Sie diese Wahrscheinlichkeitsverteilung mit der entsprechenden Methode `Math.random()` um und senden Sie, dem jeweilige Resultat entsprechendm, eine `AttackMessage` an den betroffenen Spieler.

SHOT

- Diese Nachricht wird gesendet, wenn ein Spieler einen Schuss abgeben will. Die Nachricht enthält die x und y Koordinaten des ausgewählten Feldes. Der `ClientHandler` muss nun die ausgewählte Position mit der `opponentMap` vergleichen und das Resultat in Form einer `ResultMessage` an den Spieler und den Gegner senden. Hinweis: Hierzu muss die Methode `sendMessage` für die aktuelle `ClientHandler` Instanz und die `ClientHandler` Instanz des Gegners (welche durch das Attribut `opponent` referenziert wird) aufgerufen werden.
- Beachten Sie, dass als Parameter des Konstruktors der `ResultMessage` definiert werden muss, ob der Schuss vom Gegner kam.
- Setzen Sie danach unabhängig von dem Ergebnis die getroffene Stelle in der `opponentMap` auf 0.
- Nach jedem abgefeuerten Schuss muss außerdem berechnet werden, ob der letzte Schuss zum Sieg geführt hat. Verwenden Sie hierfür die Methode `calculateVictory()`. Ist dies der Fall, senden Sie eine `WinMessage` an beide Spieler und entfernen danach den Zeiger zum Gegner (durch das Zurücksetzen des Attributs `opponent`).
- Wenn das Spiel nicht zu Ende ist, ist der Gegner am Zug. Senden Sie dem Gegenspieler dafür eine `AttackMessage`.

Aufgabe 2: Client.ServerListener implementieren (12.5 P)

In dieser Aufgabe müssen sie die Kommunikationsfähigkeit des Clients implementieren. Vervollständigen Sie hierzu die Methoden `sendMessage()` und `task2()` in der Klasse `ServerListener`, welche eine innere Klasse der Klasse `Client` ist, wie folgt:

- Vervollständigen sie die Methode `sendMessage()` so, dass sie die als Parameter übergebene Nachricht an den Server sendet.
- In der Methode `task2()` (die von der Methode `transact` aufgerufen wird), soll über den `ObjectInputStream input` ein Objekt eingelesen werden. Casten sie dieses Objekt auf die Klasse `Message`. Sie können davon ausgehen, dass nur `Message`-Objekte gesendet werden dürfen.
- Es gibt mehrere Arten von `Message`-Objekten, die jeweils durch ihren `MessageType` unterschieden werden. Ergänzen Sie nun in der Methode `task2()` die Verarbeitung der verschiedenen Nachrichten und die zugehörigen Antworten. Casten Sie dabei das `Message`-Objekt jeweils auf die entsprechende Nachrichten-Klasse und verwenden Sie ggf. die Daten, die diese Nachricht transportiert.

  START

  – Diese Nachricht signalisiert dem Spieler, dass ein Match(-Partner) gefunden wurde. Speichern Sie zunächst den Namen des Gegners im Client (im Attribut `opponentName`) und wechseln sie mit der Methode `switchToGame()` der Klasse `Window` zu dem Spielgeschehen.

  QUIT

  – Wenn diese Nachricht empfangen wurde, bedeutet dies, dass der Gegner das Spiel verlassen hat. Informieren Sie den Spieler mit einem passenden Pop-up-Fenster (mit Hilfe einer passenden Methode aus der Klasse `JOptionPane`) und wechseln Sie danach mit der Methode `switchToMenu` der Klasse `Window` zu dem Hauptmenü. Stellen Sie dabei über den Parameter der Methode sicher, dass der Spieler zum Startmenü und nicht zum Wartemenü geleitet wird.

  ATTACK

  – Diese Nachricht wird empfangen, wenn der Spieler zur Angriffsphase wechseln soll. Setzen Sie dafür die Phase der `GameScene` auf `FIGHTING`.

  – Informieren Sie danach den Spieler über den Beginn seines Zuges. Verwenden Sie hierfür die Methode `setNotification()` der Klasse `GameScene`.

  RESULT

  – Die Nachricht `ResultMessage` ist die Antwort des Servers auf einen Schuss von einem der beiden Spieler, welche mögliche Treffer oder Fehlschüsse beinhaltet. Holen Sie sich aus der Nachtricht zunächst die Informationen über die Position des Schusses und ob es sich um einen Treffer handelt.

– Je nachdem, ob der Schuss vom Spieler oder vom Gegner kam, speichern Sie das Ergebnis mit der Methode `saveResult()` der Klasse `Field` in dem zugehörigen Feld (vom Spieler oder vom Gegner) ab. Den Zugriff auf die jeweiligen Spielfelder erhalten sie über die Methoden `getPlayerField()` und `getEnemyField()` der Klasse `GameBoard`.

  Hinweis: Den Zugriff auf die benötigte Instanz der Klasse `GameBoard` erhalten Sie über `GameScene.getGameScene().getGameBoard()`.

– Wenn ein Treffer vorliegt, soll dieser auch visuell dargestellt werden. Rufen Sie hierür die Methode `shakeScreen()` der Klasse `GameScene` auf. Damit es zu keinen visuellen Fehlern kommt, malen sie zum Schluss das `Window` neu.

WIN

– Diese Nachricht wird empfangen, wenn ein Spieler das Spiel gewonnen hat. überprüfen sie hierfür zuerst, wer der Sieger ist und geben sie über ein Pop-up-Fenster eine entsprechende Nachricht aus sowie die Frage, ob der Spieler eine weitere Runde spielen will, (Die Überschrift des Fensters soll für den Sieger „You Win“ und für den Verlierer „You Lost“ sein). Für das Pop-up-Fenster sollten Sie wieder eine passende Methode aus der Klasse `JOptionPane` verwenden (mit einer `YES_NO_OPTION`).

– Wenn der Spieler mit „JA“ (bzw. „YES“) antwortet, soll er direkt mit der Methode `switchToMenu()` in das Wartemenü weitergeleitet werden. Hierfür soll an den Server eine Nachricht des Typs `Ready` mit den entsprechenden Parametern gesendet werden.

– Drückt der Spieler „Nein“ (bzw. „No“), soll er in das Hauptmenü weitergeleitet werden. Hierzu muss an den Server eine Nachricht des Typs `Cancel` gesendet werden.

Aufgabe 3: `MatchCreator` implementieren (5 P)

In dieser Aufgabe soll die Methode `findWaitingClients()` der (inneren) Klasse `server.MatchCreator` implementiert werden. Diese Methode wird über die run-Methode der Klasse jede Sekunde einmal aufgerufen und hat die Aufgabe, zwei Spieler, die aktiv nach einem Match suchen, zu identifizieren und sie anschließend in ein gemeinsames Spiel zu setzen. Gehen Sie hierzu wie folgt vor:

- Das Attribut `allClients` stellt eine Liste dar, welche alle Instanzen der Klasse `ClientHandler` enthält, die momentan mit dem Server verbunden sind. Iterieren Sie über diese, bis sie einen Client finden, welcher nicht beschäftigt (`Occupied`) ist. Merken Sie sich die (ersten beiden) gefundenen Clients mit Hilfe einer lokalen Variablen vom Typ `List`.

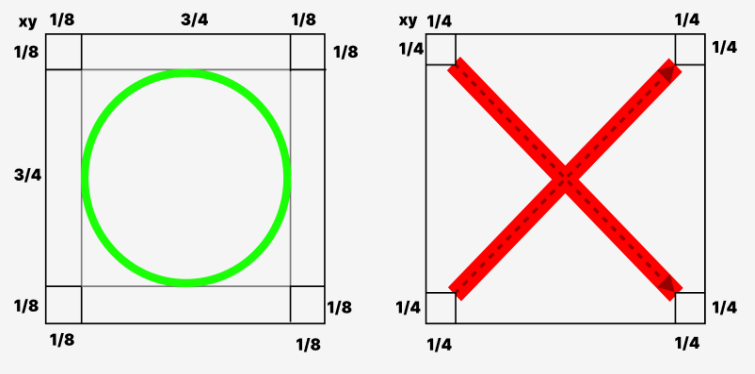
- Sobald Sie zwei Clients gefunden haben, brechen sie die Suche ab und starten sie das Match. Setzen Sie dafür beide `ClientHandler` auf beschäftigt und setzen Sie mit der Methode `setOpponent()` jeweils den anderen als Gegner.
- Senden Sie nun noch an die Clients eine `Start` Nachricht mit dem jeweils dazugehörigen Namen des Gegners.
- Stellen Sie sicher, dass es zu keinem Fehler kommen kann, der durch einen gleichzeitigen Zugriff mehrerer Threads auf die Liste `allClients` kommen kann.

Aufgabe 4: Treffer darstellen (4 P)

In dieser Aufgabe fügen sie die Markierungen für Treffer und Fehlschüsse den einzelnen Feldern hinzu. Vervollständigen sie hierfür die Methode `paintSymbol` der Klasse `Field` wie folgt:

- überprüfen sie zunächst, in welchem Zustand (`condition`) sich das Feld befindet.
- Wenn das Feld getroffen wurde (`HIT`), zeichnen sie ein Rotes Kreuz in die Mitte des Feldes mithilfe des mit der Methode übergebenen `Graphics`-Objektes.
- Wurde das Ziel verfehlt (`Miss`), soll ein grüner Kreis in die Mitte des Feldes gezeichnet werden.
- Wenn noch kein Schuss abgegeben wurde (`NONE`), soll nichts gezeichnet werden.

Die folgenden Grafiken veranschaulichen das Resultat. Sie müssen nur die roten bzw. grünen Figuren zeichnen. Die relativen x und y Werte der Nullkoordinate sind in der Methode bereits vorgegeben, sowie das `Graphics2D`-Objekt, mit dem sie die Figuren zeichnen sollen. Behalten Sie das Seitenverhältnis bei, sowie den Abstand zu dem Rand. Gehen Sie davon aus, dass das Feld immer quadratisch ist und verwenden Sie wie vorgegeben für die Strichstärke die Größe 3. **Hinweis:** Mit 1/8 bzw. 1/4 ist ein Achtel bzw. ein Viertel der Höhe gemeint.



*Bonusaufgabe: eigenes Schiff hinzufügen (3 P)

In manchen Variationen des Spiels gibt es weitere Arten von Schiffen. Fügen Sie ihr eigenes hinzu, welches eine andere Größe als die bisher vorgegebenen Schiffe hat.

- Zeichnen sie hierfür zunächst ihr eigenes Schiff in einer passenden Größe (ein Feld entspricht 10 × 10 Pixel, also müssen Länge und Breite ein Vielfaches von 10 Pixeln sein), speichern Sie es im PNG Format und und fügen sie es dem Projekt-Unterverzeichnis `resources/ships/` hinzu.
- Gehen sie nun in das Enum `ShipType` und fügen sie den Namen ihres Schiffes als Konstante hinzu.
- Passen Sie nun noch die Methoden `getSize()`, `getRelativeSize()` und `getIcon()` mit den entsprechenden Parametern an. Halten Sie sich hierbei zur Orientierung an die bereits vorhandenen Schiffe.

Das Programm erkennt automatisch, dass ein neues Schiff existiert und fügt es selbständig zum Spiel hinzu.