Ein fesselndes Multiplayer-Spiel auf P2P-Basis.

Semesterarbeit

Titel: MultiSweeper
Ein fesselndes Multiplayer-Spiel auf P2P-Basis

Autor: Bernhard Stähli

Betreuer: Keno Albrecht

Professor: Prof. Roger Wattenhofer

Forschungsgruppe: Distributed Computing Group

Datum: 14. September 2004

1. Zusammenfassung

Das Ziel dieser Semesterarbeit war, aus dem bekannten Spiel Minesweeper eine Multiplayer-Version zu entwickeln. Als Grundlage diente das P2P-System Clippee, an dem MultisweeperClients andocken können. Das Spiel wird auf diesen MultisweeperClients ausgeführt. Clippee verwaltet laufende Spiele, damit neue Spieler laufenden Spielen finden und ihnen beitreten können. Der Schwerpunkt der Arbeit lag in der Entwicklung und Implementation von geeigneten Protokollen, die die Kommunikation zwischen Clippee und MultisweeperClients sowie zwischen den MultisweeperClients selbst ermöglichen. Daneben musste ein geeignetes Benutzerinterface für die MultisweeperClients entwickelt werden, um das Spielbrett darzustellen. Die besondere Herausforderung bestand darin, die Probleme der verteilten Programmierung zu lösen.

2. Einführung

Minesweeper ist ein beliebtes und verbreitetes Spiel. Leider ist es nur in einer Einspieler-Version implementiert. Deshalb ist die Idee entstanden, eine Multiplayer-Version des Spiels zu entwickeln. Die Arbeit wurde am Institut für Distributed Computing der ETH Zürich ausgeschrieben, um eine Anwendung für das P2P-System Clippee zu erstellen.

Das Spiel wird Multisweeper genannt und ist in Java realisiert. Multisweeper baut auf dem bestehenden P2P-System Clippee auf. Clippee ist so konstruiert, dass daran Clients andocken können, um Informationen abzufragen. Die Clients wurden für Multisweeper zu MultisweeperClients erweitert. Sie beherbergen das Spielbrett und die Netzwerkfunktionalität, um mit Clippee und anderen MultisweeperClients zu kommunizieren.

Einem MultisweeperClient ist es möglich, über Clippee neue Spiele zu gründen, laufende Spiele ausfindig zu machen und Spielen beizutreten. Einem Spiel können während seiner ganzen Laufzeit neue Spieler beitreten.

3. Aufgabenstellung

Multiplayer-Netzwerk-Spiele sind sehr populär. In dieser Semesterarbeit soll eine Multiplayer-Version von Minesweeper erstellt werden. Es sollen die Regeln dahingehend erweitert werden, dass das Spiel mehrspielerfähig wird. Alle Spieler sollen gleichzeitig dasselbe Minenfeld bearbeiten. Um die Kommunikation zwischen den Spielern zu ermöglichen, soll das Spiel in ein bestehendes P2P-System namens Clippee integriert werden. Das Verstehen dieses Frameworks ist ein Bestandteil der Arbeit. Das Ziel der Arbeit ist das Verstehen der Probleme, die sich bei verteilten Anwendungen und im Speziellen bei “Real-Time“-Games ergeben.

4. Spielregeln

Multisweeper ist, wie der Name schon sagt, ein Multiplayer-Game. Es geht darum, mehr Punkte zu sammeln als die Mitspieler. Wer am Ende des Spiels die meisten Punkte erreicht hat, ist der Sieger. Das Spielfeld besteht aus einer Menge von kleinen verdeckten Feldern. Hinter jedem bedeckten Feld kann sich entweder ein freies Feld oder eine Mine befinden. Ziel ist es, die Minen zu erkennen und die freien Felder aufzudecken. Jeder Spieler, der ein freies Feld aufdeckt, erhält einen Punkt. Es wird nur ein Punkt vergeben, auch wenn durch das Klicken eines freien Feldes automatisch weitere freie Felder aufgedeckt werden.

Klickt ein Spieler versehentlich auf eine Mine, so wird seine Punktzahl auf -5 zurückgesetzt. Ist seine Punktezahl bereits negativ, so wird die Punktezahl nach dem Klicken auf eine Mine um zusätzliche 5 Punkte reduziert. Wenn zwei Spieler quasi gleichzeitig ein freies Feld aufdecken, wird der Punkt beiden Spielern vergeben. Es ist also möglich, dass die Summe der Punktzahlen aller Spieler über der Anzahl der freien Felder des Spielfeldes liegt.

5. Spielablauf

5.1. Spielstart

Wenn ein Spieler Multisweeper spielen möchte, muss er einen MultisweeperClient starten, der sich dann bei Clippee anmeldet.

Um mit dem Spielen zu beginnen, werden immer folgende Daten benötigt:

- **Playername**:
  Der Name des Spielers, der auf dem Spielbrett angezeigt wird.
- **Clipee IP**:
  Die IP-Adresse eines gestarteten MultisweeperPeers.
- **Clippee Port**:
  Der Port, auf dem der MultisweeperPeer auf eingehende Verbindungen von MultisweeperClients wartet. Dies ist die Portnummer `<peerPort>+1`. Der Wert `<peerPort>` wurde beim Starten des MultisweeperPeers festgelegt.
- **Listening Port**:
  Der Port, auf dem ein MultisweeperClient auf Verbindungen von anderen MultisweeperClients wartet, die einem laufenden Spiel beitreten möchten.

Es gibt zwei Möglichkeiten, um mit dem Spielen zu beginnen. Einerseits kann ein neues Spiel gestartet werden, und andererseits kann einem bestehenden Spiel beigetreten werden.

5.1.1. Neues Spiel starten

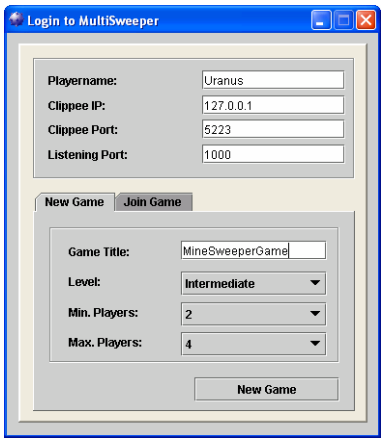


Abbildung 1: Login Screen: Neues Spiel erstellen

Um ein neues Spiel zu starten, müssen zusätzliche Daten angegeben werden:

- **Game Title:**
  Der Titel des Spiels. Dieser wird angezeigt, wenn andere Spieler nach laufenden Spielen suchen.
- **Level:**
  Gibt die Grösse des Spielfeldes an.
  Beginner: Spielbrett mit 8 x 8 Feldern.
  Intermediate: Spielbrett mit 16 x 16 Feldern.
  Expert: Spielbrett mit 30 x 16 Feldern.
- **Min. Players:**
  Gibt die minimale Anzahl Spieler an. Das heisst, Multisweeper wartet mit dem Starten des Spiels, bis diese minimale Anzahl von Spielern dem Spiel beigetreten ist. Erst dann wird das Spiel für alle gleichzeitig gestartet.
- **Max. Players:**
  Gibt an, wie viele Spieler dem Spiel maximal beitreten können.

5.1.2. Einem bestehenden Spiel beitreten

Um einem bestehenden Spiel beizutreten, muss ein laufendes Spiel gesucht werden. Dies geschieht, indem man auf die Schaltfläche „Refresh“ klickt. Es werden alle Spiele angezeigt, die von Clippee verwaltet werden.

Zu jedem Spiel können die Spieleigenschaften angezeigt werden, indem man den Tree-Node des entsprechenden Spiels erweitert.

Um ein Spiel auszuwählen, wird es angeklickt. Sobald das gewünschte Spiel ausgewählt ist, kann man durch Anklicken der Schaltfläche „Join Game“ dem Spiel beitreten.

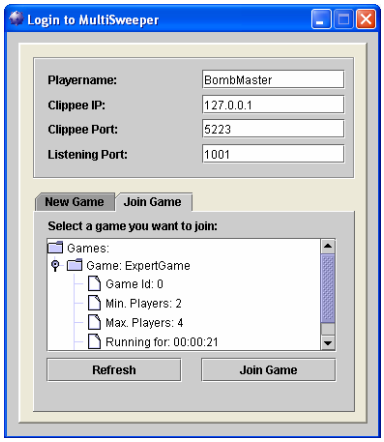


Abbildung 2:
Login Dialog: Join Game

5.1.3. Aufstarten des Spiels

Wenn ein neues Spiel erstellt wird, so erscheint ein Dialogfeld, welches anzeigt, dass auf die verbleibende minimale Anzahl von Spielern gewartet werden muss. Sind dem Spiel genügend Spieler beigetreten, so gelangt man zum Spielfeld.
Derselbe Dialog wird auch angezeigt, wenn man einem Spiel beitritt, dem auch nach dem eigenen Beitreten noch nicht genügend Mitspieler beigetreten sind.

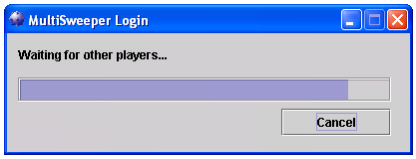


Abbildung 3:
Login Dialog: Join Game

5.2. Spielverlauf

Sobald sich genügend Spieler eingefunden haben und das Spiel aufgestartet ist, wird das Spielbrett angezeigt.

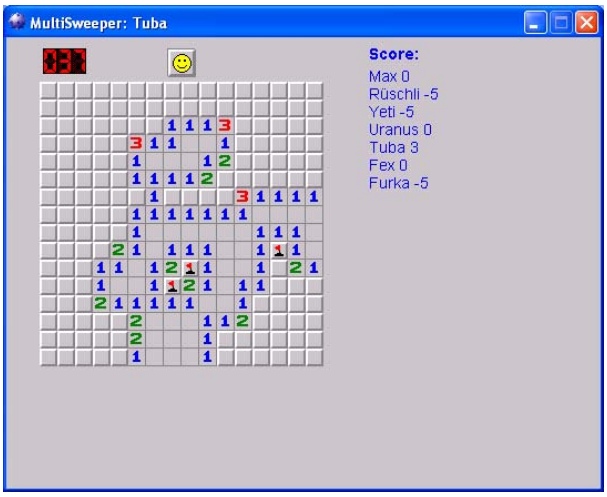


Abbildung 4: Multisweeper Spielbrett

Um den Spieler der den ersten Klick macht nicht zu benachteiligen, sind bereits einige Felder aufgedeckt. Rechts werden die Namen der am Spiel beteiligten Mitspieler zusammen mit ihrer momentanen Punktezahl angezeigt. Oben links sieht man, wie viele Minen verdeckt sind, vorausgesetzt die selbst gesetzten Flags sind alle richtig gesetzt.

5.2.1. Spielsteuerung

Um ein Feld aufzudecken, muss mit der linken Maustaste auf das verdeckte Feld geklickt werden. Wenn man hinter einem Feld eine Mine vermutet, so kann man das Feld mit einem Flag markieren. Dies wird erreicht, indem man mit der rechten Maustaste das verdeckte Feld klickt. Um ein Flag wieder zu entfernen, klickt man es erneut mit der rechten Maustaste an.

Wenn man sehen möchte, wo andere Mitspieler ihre Flags gesetzt haben, so drückt man die Ctrl-Taste. Während dem die Ctrl-Taste gedrückt ist, werden die Flags der Mitspieler in blauer Farbe dargestellt.

Wird ein Feld angeklickt und ist keine Mine darunter versteckt, so wird das Feld aufgedeckt. Für jedes aufgedeckte Feld erhält der Spieler einen Punkt. Wird aber ein Feld angeklickt, unter dem sich eine Mine befindet, so werden die Punkte des Spielers auf -5 zurückgesetzt. War seine Punktezahl negativ, als er die Mine angeklickt hat, so werden der bisherigen negativen Punktzahl zusätzliche 5 Punkte abgezogen. Einige der aufgedeckten Felder enthalten Nummern. Diese geben an, wie viele Minen sich in den umliegenden 8 Feldern befinden. Steht eine 3 in dem Feld, dann bedeutet dies, dass in den angrenzenden 8 Feldern genau 3 Minen versteckt sind.

Wird ein freies Feld aufgedeckt, so ist dies sofort für alle Mitspieler sichtbar. Wird aber eine Mine angeklickt, so wird dieses Feld nur lokal aufgedeckt. Es wird eine rote Mine angezeigt. Die anderen Mitspieler können dieselbe Mine auf ihrem eigenen Spielbrett immer noch anklicken.

5.2.2. Spielende

Das Spiel ist beendet, wenn alle Felder aufgedeckt sind, unter denen keine Mine versteckt ist. Das Spiel ist ebenfalls beendet, wenn, bis auf einen Spieler, alle das Spiel verlassen haben. Nach dem das Spiel beendet ist, wird eine Rangliste angezeigt. Der Spieler, der die höchste

Punktzahl erreicht hat, ist der Sieger. Dies ist der Spieler, der am meisten freie Felder aufgedeckt hat und dabei möglichst wenig Minen erwischt hat.

6. Clippee und MultisweeperPeers

6.1. Clippee

6.1.1. Aufbau von Clippee

Der Clippee ist eine Menge von MultisweeperPeers, die untereinander verbunden sind. Der Verbund kann aus 1 bis n MultisweeperPeers bestehen. Die MultisweeperPeers sind durch einen vollständigen Graphen miteinander verbunden. Das heisst, es handelt sich um eine P2P-Topologie. Durch den Zusammenschluss verschiedener MultisweeperPeers ist eine Verteilung der Aufgaben möglich, so dass eine gute Skalierbarkeit und Performance erreicht werden kann. Um Informationen abzurufen, können an Clippee Clients andocken. Die Clients, die Multisweeper verwendet, werden im Folgenden MultisweeperClients genannt.

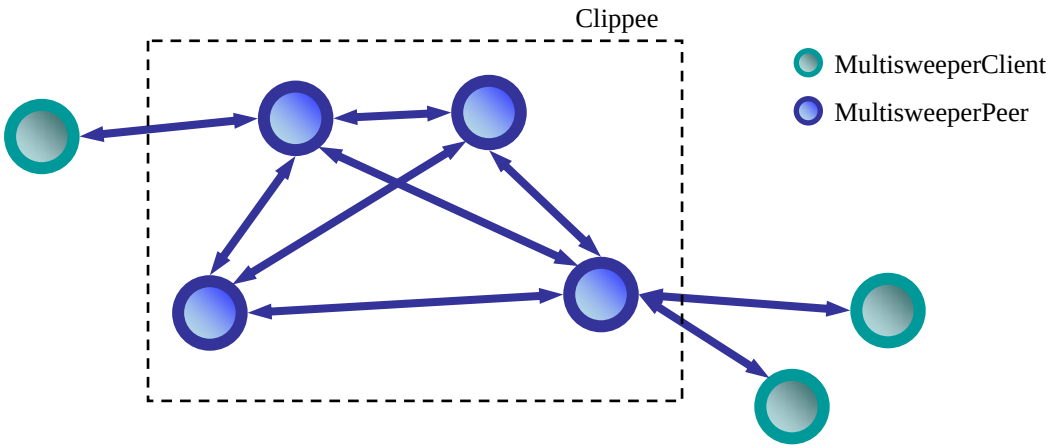


Abbildung 5: Clippee mit angedockten MultisweeperClients

6.1.2. Aufgaben von Clippee

In erster Linie hat Clippee die Aufgabe, laufende Spiele und die dazu gehörenden Spieler zu verwalten. Sobald ein neues Spiel gestartet wird, wird dieses in Clippee abgespeichert. Wenn später ein MultisweeperClient an Clippee andockt, kann er alle in Clippee gespeicherten Spiele abfragen und einem dieser Spiele beitreten. Um dem Spiel beizutreten, erhält der neue MultisweeperClient von Clippee alle benötigten Daten.

6.2. MultisweeperPeers

Clippee ist ein P2P-System und setzt sich aus einzelnen MultisweeperPeers zusammen. Jeder MultisweeperPeer ist identisch und hat die selben Fähigkeiten.

Im Wesentlichen ist die Funktionalität der MultisweeperPeers in drei Teile gegliedert. In jedem MultisweeperPeer läuft je eine Instanz eines ClippeePeers, eines LoginPeers und eines GamePeers.

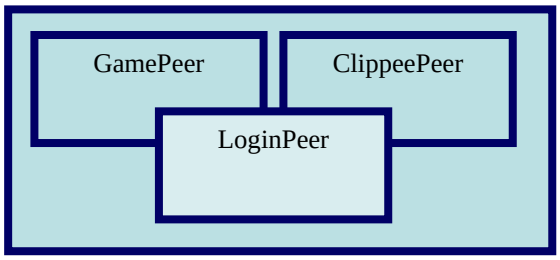


Abbildung 6: Aufbau des MultisweeperPeers

6.2.1. ClippeePeer

Der ClippeePeer hat die Aufgabe, die MultisweeperPeers miteinander zu verbinden. Es können neue Multisweeper Peers zum Verbund hinzugefügt und später wieder entfernt werden. Der ClippeePeer kann Nachrichten an alle MultisweeperPeers von Clippee senden.

6.2.2. LoginPeer

Der LoginPeer ermöglicht es, dass MultisweeperClients an einen MultisweeperPeer andocken, um bestimmte Funktionen auszuführen. Der Client kann zum Beispiel die in Clippee gespeicherten Spiele abfragen, ein neues Spiel erstellen oder einem bestehenden Spiel beitreten.

6.2.3. GamePeer

Der GamePeer verwaltet die Spiele, die über den MultisweeperPeer laufen, zu dem der GamePeer gehört. Der GamePeer koordiniert beim Starten des Spiels das Warten auf die minimale Spieleranzahl.

6.3. Bedienung von Clippee und der MultisweeperPeers

Beim Start eines MultisweeperPeers gibt es die Möglichkeit, ein neues Clippee zu gründen oder sich in ein bestehendes Clippee zu integrieren.

6.3.1. Ein neues Clippee gründen

Wenn ein neues Clippee gegründet werden soll, so muss lediglich der Port angegeben werden, unter welchem andere MultisweeperPeers dieses Clippee erreichen können, um sich selbst in dieses zu integrieren. Der entsprechende Parameter heisst `<peerPort>`.

Befehl:

```
java –jar MultiSweeperPeer.jar <peerPort>
```

6.3.2. Einen neuen MultisweeperPeer in ein bestehendes Clippee integrieren

Besteht bereits ein Clippee, kann man dieses mit weiteren MultisweeperPeers erweitern, um zum Beispiel die Verfügbarkeit zu erhöhen. In diesem Fall muss man die IP-Adresse und den Listening-Port eines beliebigen MultisweeperPeers, der Bestandteil des gewünschten Clippees ist, kennen. Um sich in ein Clippee zu integrieren, gibt man also die IP-Adresse `<connectIP>` und zusätzlich den Port `<connectPort>` an, auf dem ein MultisweeperPeer läuft, der Bestandteil von Clippee ist. Wie im oberen Fall muss auch hier der Port `<peerPort>` angegeben wer-

den, auf dem der neue MultisweeperPeer auf eingehende Verbindungen von anderen MultisweeperPeers wartet, die sich bei Clippee anmelden wollen.

Befehl:
```
java –jar MultiSweeperPeer.jar <peerPort> [<connectIP> <connectPort>]
```

7. MultisweeperGames und MultisweeperClients

7.1. Aufbau eines MultisweeperGames

Ein MultisweeperGame ist ein P2P-Netzwerk. Ähnlich wie in Clippee bilden verschiedene Peers ein Netz. Die Peers heissen im MultisweeperGame MultisweeperClients. Aus Sicht von Clippee sind die MultisweeperClients Clients, die andocken können. Aus Sicht des MultisweeperGames sind die beteiligten MultisweeperClients Peers, die ein P2P-Netzwerk bilden. Jeder MultisweeperClient des MultisweeperGames ist mit dem MultisweeperPeer verbunden, der das Spiel verwaltet. Dieser MultisweeperPeer ist wiederum Bestandteil von Clippee.

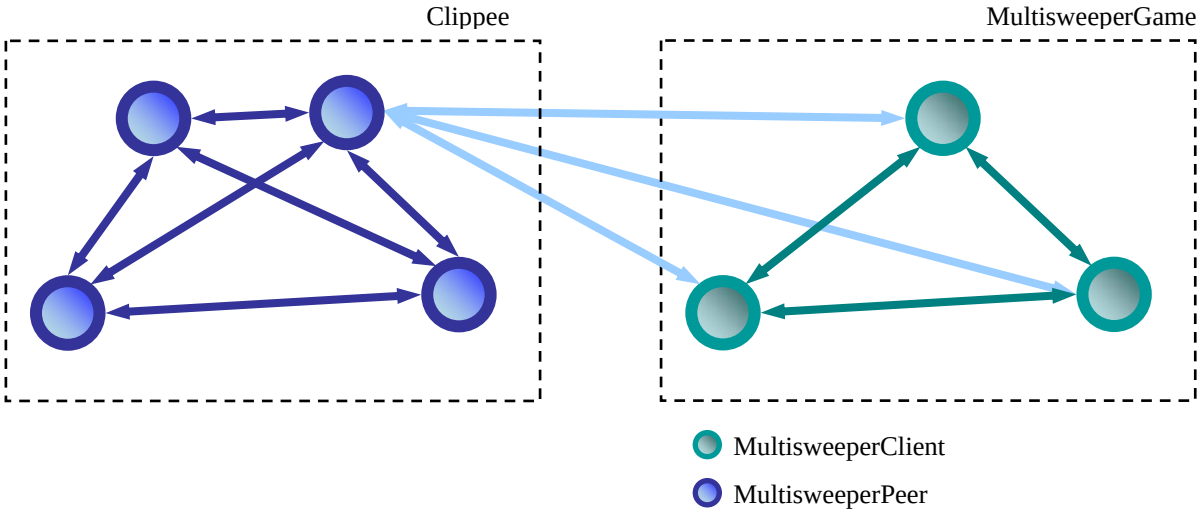


Abbildung 7: Topologie von Clippee zusammen mit einem MultisweeperGame

7.2. Starten des MultisweeperClients

Zum Starten des MultisweeperClients sind keine Argumente nötig. Die benötigten Informationen werden über das User-Interface, das gleich nach dem Start angezeigt wird, angegeben.

Zum Starten eines MultisweeperClients gibt man folgenden Befehl ein:

```
java –jar MultiSweeper.jar
```

7.3. Aufbau des MultisweeperClients

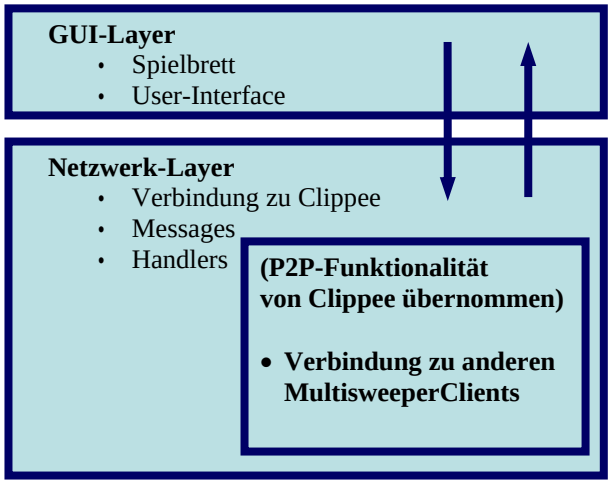


Abbildung 8: Aufbau des MultisweeperClients

Der MultisweeperClient ist in zwei Schichten aufgeteilt. Es gibt einen GUI-Layer und den Netzwerk-Layer.

7.3.1. Der GUI-Layer

Der GUI-Layer enthält verschiedene Klassen zur Darstellung von Dialogen und Fenstern. Es werden die Klassen von `javax.swing.*` verwendet, um das User-Interface darzustellen. Die Klassen des GUI-Layers sind im Package `multisweeper.gui` enthalten. Die wichtigste Klasse ist `MultisweeperGamePanel`. Diese Klasse stellt das Spielbrett dar und reagiert auf die Eingaben des Spielers. Das Spielbrett muss aber auch auf dem aktuellen Stand gehalten werden. Das heisst, es werden Funktionen bereitgestellt, die es der Netzwerkschicht erlauben, das Spielbrett zu aktualisieren. Dies ist zum Beispiel der Fall, wenn ein Spieler ein Feld aufdeckt. Nachdem er das Feld aufgedeckt hat, teilt er dies all seinen Mitspielern mittels einer `UncoverMessage` mit. Die Netzwerkschicht löst beim Erhalt der `UncoverMessage` ein Update des Spielbrettes aus, so dass wieder alle Spieler eine konsistente Sicht auf das Spielbrett haben.

Schnittstelle Netzwerk-Layer – GUI-Layer
Nachfolgend sind die wichtigsten Funktionen der Klasse `MultisweeperGamePanel` aufgelistet, die die Schnittstelle Netzwerk-Layer – GUI-Layer bilden:

```
public void setRemoteFlag(int i, int j, int index, long playerId)
```
Ein Mitspieler mit der Id `playerId` hat auf dem Feld `(i, j)` ein Flag gesetzt oder entfernt. Die Variable `index` gibt an, ob ein Flag gesetzt oder entfernt wurde.

```
public void setGameWorld(GameWorldObject myGameWorld)
```
Der Spieler, der das neue Spiel gegründet hat, stellt den anderen Mitspielern, die dem Spiel beitreten, ein `GameWorldObject` zur Verfügung. Dieses Objekt enthält alle Informationen, um ein Spielbrett aufzubauen. Die Funktion `setGameWorld()` nimmt ein solches `GameWorldObject` entgegen und baut das eigene Spielbrett entsprechend auf.

```
public void removePlayer(long remotePlayerId)
```

Wenn ein Spieler das Spiel vorzeitig verlässt, so muss er aus der Rangliste entfernt werden. Der Funktion `removePlayer()`entfernt den Spieler mit der Id `remotePlayerId` aus der Rangliste.

```
public GameWorldObject getGameWorld()
```
Um bei den beitretenden Spielern die Funktion `setGameWorld()` aufzurufen, muss zuerst ein `GameWorldObject` erstellt werden. Die Funktion `getGameWorld` sammelt alle relevanten Daten des Spielbrettes und speichert sie in einem `GameWorldObject` ab.

```
public void clickAt(int i, int j, long remotePlayerId, boolean wonGame)
```
Die Funktion `clickAt()` wird aufgerufen, wenn ein Mitspieler ein Feld aufdeckt. Das Feld `(i, j)` wird auf dem Spielbrett aufgedeckt, und dem Spieler mit der Id `remotePlayerIdInt` wird ein Punkt gutgeschrieben.

```
public void addPlayer(String remotePlayerId, long remotePlayerId, Connection connection)
```
Wenn ein neuer Spieler dem Spiel beitritt, so werden die Spielbretter aller bereits beteiligten Spieler informiert, damit sie den neuen Spieler in die Rangliste aufnehmen können.

```
public void resetPoints(long remotePlayerIdInt)
```
Wenn der Spieler mit der Id `remotePlayerIdInt` auf eine Mine klickt, so werden die Punkte dieses Spielers zurückgesetzt.

7.3.2. Der Netzwerk-Layer

Der Netzwerk-Layer hat die Aufgabe die verschiedenen Peers miteinander zu verbinden. Die Peers werden durch Verbindungen zusammengehalten. Ein MultiSweeperGame ist ein P2P-Netzwerk, wobei die Peers durch einen vollständigen Graphen verbunden sind. Die Mitspieler-MultisweeperClients eines MultisweeperClients werden in einer Klasse `PeerManager` verwaltet. Eine wichtige Funktion des PeerManagers ist `sendAllPeers(final Message message)`. Mit ihr kann man eine Broadcast-Nachricht an alle Peers eines P2P-Netzwerks schicken. Diese wird in MultiSweeper sehr oft verwendet, da eine Änderung des Spielbrettes immer allen Mitspielern mitgeteilt werden muss.

Die P2P-Funktionalität, die ich für den Aufbau des MultisweeperGame-P2P-Netzwerk benötigt habe, konnte ich weitgehend aus Clippee übernehmen. Sie bildet die Grundlage für die Kommunikation zwischen den MultisweeperClients. Eine funktionierende Kommunikation ermöglicht wiederum die Implementierung von komplexeren Protokollen. Wie die Kommunikation im Einzelnen Abläuft, ist im Kapitel Kommunikation weiter unten beschrieben.

8. Kommunikation

8.1. Kommunikation mit Messages und Connections

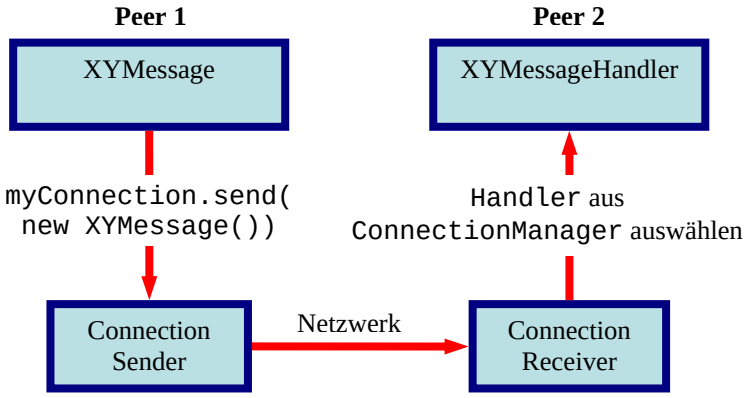


Abbildung 9: Versand einer XYMessage

8.1.1. Die Klasse Message

Die Klasse `Message` ist ein Objekt, welches über eine `Connection` verschickt werden kann. Das besondere an einem `Message`-Objekt ist, dass man es serialisieren kann. Das heisst, das `Message` Objekt kann man in einen Bytestrom verwandeln, über ein Netzwerk senden und beim Empfänger wieder zu einem `Message`-Objekt zusammenbauen.

Um Daten zu verschicken, kann eine `Message` `Parameter` aufnehmen. Mit der Funktion `addParameter(Parameter parameter)` kann der Massage ein `Parameter` hinzugefügt werden. Parameter haben einen bestimmten Typ und sind von der Klasse `Parameter` abgeleitet. Ein `IntegerParameter` zum Beispiel kann eine Integer-Variable aufnehmen.

Wenn man in einer Nachricht einen Spielernamen zusammen mit seiner Punktezahl verschickten will, erstellt man zuerst eine `Message`, fügt ihr dann einen `StringParameter` für den Spielernamen und anschliessend einen `IntegerParameter` für die Punktezahl hinzu.

8.1.2. Objekt-Nachrichten

Wenn man ein Objekt versenden will, so muss man die Attribute des Objektes zuerst einzeln in passende `Parameter` verpacken, um diese dann der `Message` hinzuzufügen. Beim Empfänger angekommen muss das versandte Objekt aus den Parametern wieder aufgebaut werden. Dies erfordert einen grossen Programmieraufwand und ist sehr fehleranfällig. Insbesondere muss immer die Sender und die Empfängerklasse angepasst werden, wenn man das zu versendende Objekt nachträglich abändert.

Um diesen Prozess zu vereinfachen, habe ich eine von der Klasse `Message` abgeleitete Klasse `ObjectMessage` geschrieben, die nur einen einzigen `Parameter` enthält – eine von der Klasse `Parameter` abgeleiteten Klasse namens `SerializedObjectParameter`. Um eine `ObjectMessage` zu erstellen, reicht es, dem Konstruktor ein serialisierbares Objekt zu übergeben.

```
ObjectMessage(Serializable myObject, byte TYPE)
```

Die erstellte Nachricht ist dann automatisch serialisierbar, da das angegebene Objekt im Konstruktor der `ObjetMessage` in einen `SerializedObjectParameter` verwandelt wird, der das Serialisieren und Deserialisieren des Objektes übernimmt. Somit wird die Implementierung von Protokollen einfacher, da man die zu versendende Information einfach in ein Objekt packen kann, um dieses anschliessend zu versenden. Der Empfänger hat dann die ganze Information in einem Objekt gruppiert, und es wird von der eigentlichen Serialisierung abstrahiert.

Der Nachteil ist, dass solche Nachrichten einen etwas erhöhten Speicherbedarf aufweisen. Aber für Nachrichten wie die `GameWorldMessage` mit sehr vielen Attributen (es muss der Zustand jedes einzelnen Feldes des Spielbrettes übertragen werden, was über 1000 Parameter erfordern würde), ist es sehr praktisch, wenn man das gesamte `GameWorldObject` in einem Parameter verschicken kann.

8.1.3. Die Klasse Connection

Wenn ein Peer eine Nachricht einem anderen Peer senden möchte, so muss eine Verbindung zu diesem Peer existieren. Diese muss nie explizit erstellt werden, da die Topologie ein vollständiger Graph ist. Beim Hinzufügen eines Peers zum P2P-Netz wird automatisch eine Verbindung vom neuen Peer zu allen bestehenden Peers erstellt. Die Klasse `Connection` implementiert eine Funktion `sendMessage(Message message)` und eine Funktion `void sendRequest(TimeoutHandler handler)`. Dadurch ist synchrone und asynchrone Kommunikation möglich. Die Funktion `sendMessage()` sendet eine asynchrone Nachricht. Es wird keine Antwort erwartet. Die Funktion `sendRequest()` ist synchron. Im `TimeoutHandler` wird definiert, welcher Messagetyp als Antwort auf die Anfrage erwartet wird. Wird innerhalb einer definierten Zeit, dem Timeout, keine entsprechende Nachricht über die Verbindung empfangen, so wird eine im `TimeoutHandler` definierte Funktion `onTimeout(Connection connection)` aufgerufen.

Über eine Verbindung können Nachrichten verschiedener Typen verschickt werden. Damit der Empfänger weiss, was beim Empfang einer Nachricht des Typs xy zu tun ist, muss auf der Connection ein `Handler` definiert werden, der beim Empfang einer Nachricht vom Typ xy in Aktion tritt und die entsprechende Aktion durchführt.

Ein `Handler` implementiert zwei Funktionen:

```
void handle(Connection connection, Message message)
void handleRequest(Connection connection, Message message, int messageID);
```

Die Funktion `handle()` reagiert auf den Empfang einer asynchronen Nachricht. Die Funktion `handleRequest()` dagegen reagiert auf den Empfang einer synchronen Nachricht. Mit der Funktion `connection.sendAnswer(ReplyMessage, messageID)` wird auf eine synchrone Nachricht reagiert, indem eine entsprechende Reply-Nachricht zurückgesendet wird.

Für jeden Nachrichtentyp muss also ein entsprechender `Handler` erstellt und beim Empfänger auf der `Connection` registriert werden.

Der `Handler` wird auf der `Connection` mit der Funktion `registerHandler(Handler XY-Handler)`registriert.

8.2. Protokolle

Um den Ablauf des Spiels zu ermöglichen, sind eine Reihe von Protokollen nötig. Diese sind nachfolgend im Detail beschrieben.

8.2.1. New-Game-Protokoll

Das New-Game-Protokoll wird ausgeführt, wenn ein MultisweeperClient ein neues Spiel erstellen will. Er muss dazu die Netzwerkadresse eines MultisweeperPeers kennen, auf dem ein LoginPeer gestartet ist.

Als erstes wird eine `NewGameMessage` an den LoginPeer versandt. Sie enthält alle Informationen, die zum Erstellen eines neuen Spiels erforderlich sind.

MultisweeperClient ⇨ LoginPeer
`NewGameMessage(String gameName, String playerIdStr, long playerID, int maxPlayers, int minPlayers)`

Parameter:

`gameName`	Name des Spiels
`playerIdStr`	Name des Spielers in Textform
`playerId`	eindeutige Id des Spielers Dies ist ein `long`, der sich aus der IP-Adresse und dem Listening-Port des MultisweeperClients zusammensetzt.
`maxPlayers`	maximale Anzahl Spieler, die am Spiel teilnehmen können
`minPlayers`	minimale Anzahl Spieler, die benötigt wird, um das Spiel aufzustarten

Der LoginPeer wählt zwecks Lastenverteilung einen zufälligen Peer aus dem Clippee-Netzwerk aus. Dies ist der GamePeer, auf dem später das neu zu erstellende Spiel verwaltet wird.

LoginPeer ⇨ GamePeer
`GetGameMessage(String gameName, long playerID, int maxPlayers, int minPlayers)`

Parameter:

`gameName`	siehe oben
`playerId`	siehe oben
`maxPlayers`	siehe oben
`minPlayers`	siehe oben

Beim Erhalt einer `GetGameMessage` erstellt der GamePeer ein neues Spiel mit den geforderten Parametern wie `maxPlayers` und `minPlayers`, und speichert es bei sich ab. Das Spiel erhält eine eindeutige Id.

Um zu bestätigen, dass das neue Spiel erstellt wurde, schickt er eine `GotGameMessage` an den LoginPeer zurück. In der Nachricht ist die `gameId` enthalten. Diese wird später vom MultisweeperClient verwendet, um sich beim erstellten Spiel anzumelden.

GamePeer ⇨ LoginPeer
`GotGameMessage(long playerID, IPParameter gamePeerIP, int gameID)`

Parameter:

`playerId`	siehe oben
`gamePeerIp`	Der GamePeer sendet seine eigene Netzwerkadresse mit, damit sich der MultisweeperClient, der das Spiel erstellt hat, später bei ihm mit dem Join-Waiting-Game-Protokoll anmelden kann.
`gameID`	siehe oben

Zum Schluss informiert der LoginPeer den wartenden MultisweeperClient, dass das Spiel erstellt wurde und bereit ist, sich daran anzumelden.

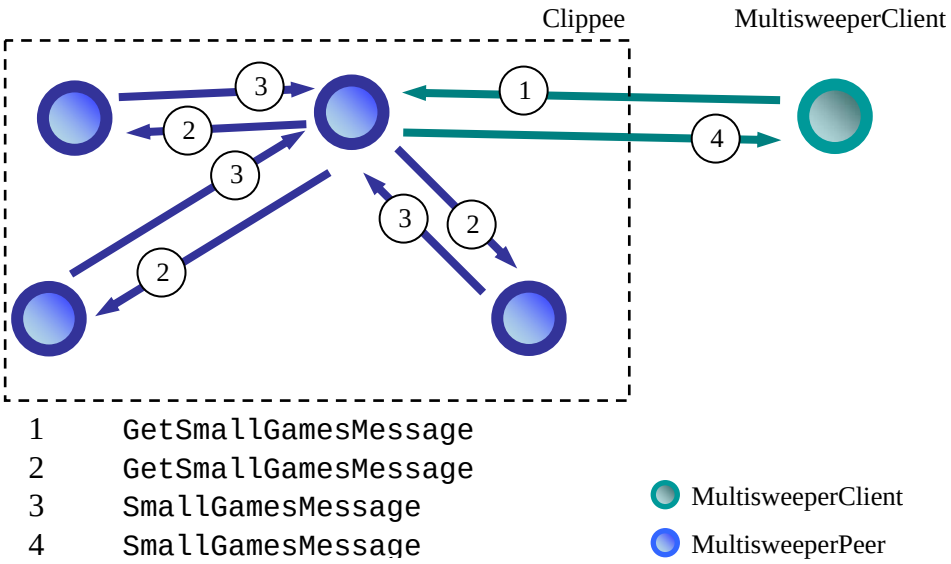
LoginPeer ⇨ MultisweeperClient
`YourGameMessage(String playerIdStr, long playerID, IPParameter gamePeerIP, int gameID)`

Parameter:

`playerIdStr`	siehe oben
`playerId`	siehe oben
`gamePeerIp`	siehe oben
`gameID`	siehe oben

Beim Erhalt einer `YourGameMessage` weiss der MultisweeperClient, dass das Spiel nun bereit ist. Er führt nun automatisch das Join-Waiting-Game-Protokoll aus, um sich als erster Spieler bei dem Spiel mit der Id `gameID` anzumelden. Dieses Protokoll wird weiter unten beschrieben.

8.2.2. Game-Discovery-Protokoll



1 `GetSmallGamesMessage`
2 `GetSmallGamesMessage`
3 `SmallGamesMessage`
4 `SmallGamesMessage`

Abbildung 10: Aufbau des MultisweeperClients

Das Game-Discovery-Protokoll dient einem MultisweeperClient dazu, bei Clippee alle verfügbaren Spiele abzufragen. Aus der erhaltenen Liste von Spielen kann er dann eines auswählen, um daran teilzunehmen.

Zuerst sendet der MultisweeperClient eine `GetSmallGamesMessage()` zu einem beliebigen MultisweeperPeer von Clippee. Dieser MultisweeperPeer wird nachfolgend MultisweeperPeer 1 genannt. Seine Netzwerk-Adresse kennt der MultisweeperClient, weil diese bei seinem Start vom Benutzer eingegeben wurde.

MultisweeperClient ⇨ MultisweeperPeer 1
`GetSmallGamesMessage()`

Wenn der MultisweeperPeer 1 eine `GetSmallGamesMessage` erhält, sendet er die Nachricht weiter an alle anderen MultisweeperPeers die Bestandteil von Clippee sind, um auch deren Spiele einzusammeln.

MultisweeperPeer 1 ⇨ alle anderen MultisweeperPeers von Clippee
`GetSmallGamesMessage()`

Beim Erhalt einer `GetSmallGamesMessage` lesen die MultisweeperPeers die eigenen laufenden Spiele aus und schicken sie in einer `SmallGamesMessage` zum MultisweeperPeer 1.

alle MultisweeperPeers von Clippee ⇨ MultisweeperPeer 1
`SmallGamesMessage(Vector mySmallGames)`

Parameter:

`mySmallGames`	Vector, der die SmallGames des MultisweeperPeers enthält

Der MultisweeperPeer 1 wartet bis alle MultisweeperPeers (innerhalb eines Timeouts) geantwortet haben. Danach fügt er alle erhaltenen `SmallGames` zusammen, hängt seine eigenen Games an und schickt die komplette Liste als Antwort an den MultisweeperClient zurück.

MultisweeperPeer 1 ⇨ MultisweeperClient
`SmallGamesMessage(Vector mySmallGames)`

Parameter:

`mySmallGames`	Vector, der alle SmallGames enthält

Der MultisweeperClient hat nun eine Liste der auf Clippee laufenden Spiele. Der Spieler kann nun sein gewünschtes Spiel auswählen, um diesem beizutreten.

SmallGames
Die Klasse `SmallGame` ist eine kompakte Version eines Games und enthält die Informationen, die benötigt werden, um einem Spiel beizutreten. Das Wichtigste ist die Netzwerk-Adresse des MultisweeperPeers, über den man dem Spiel beitreten kann.

```
private String   gameName;
private int      gameID;
private int      numberOfPlayers;
```

```
private int      minPlayers;
private int      maxPlayers;
private String   gamePeerIp;
private int      gamePeerPort;
private Vector   players;
private long     time;
```

8.2.3. Join-Waiting-Game-Protokoll

Dieses Protokoll wird ausgeführt, wenn ein Spieler einem Spiel im Warte-Zustand beitreten möchte. Ein Spiel ist im Warte-Zustand, wenn die zum Starten nötige minimale Spieleranzahl noch nicht erreicht ist.

Nach dem ein MultisweeperClient mit dem Game-Discovery-Protokoll die auf Clippee verfügbaren Spiele in Erfahrung gebracht hat, entscheidet sich der Benutzer für eines dieser Spiele, um an diesem teilzunehmen. Das Spiel wird durch seine `gameID` identifiziert. Er muss die Netzwerkadresse eines der bereits beteiligten MultisweeperClients in Erfahrung bringen, um danach das aktuelle Spielbrett bei diesem herunterzuladen.

MultisweeperClient ⇨ GamePeer
`PlayGameMessage(playerIdStr, playerID, gameID, listeningPort)`

Parameter:

`playerIdStr`	siehe oben
`playerId`	siehe oben
`gameId`	siehe oben
`listeningPort`	Port, auf dem der MultisweeperClient auf eingehende Verbindungen von MultisweeperClients wartet, die dem Spiel beitreten möchten.

Der GamePeer fügt den MultisweeperClient zum Spiel hinzu. Wenn noch nicht genügend Spieler vorhanden sind, so wird der neue Spieler in die Liste der waitingClients eingefügt. Sind dann nach einer Weile genügend Spieler angemeldet, so geht das Protokoll folgendermassen weiter: Jedem MultisweeperClient in der Liste waitingClients wird eine `Multisweeper-GameReadyMessage` zugeschickt.

GamePeer ⇨an alle waitingClients **MultisweeperClient**
`MultisweeperGameReadyMessage(String ip, int port, int gameId, Vector players)`

Parameter:

`ip`	IP-Adresse des MultisweeperClinets, von dem die GameWorld heruntergeladen werden soll. Dieser wird `King` genannt und ist der Spieler, der das Spiel aufgestartet hat.
`port`	Port des `Kings`.
`gameId`	siehe oben
`players`	Ein `Vector`, der alle Spieler-Ids und die entsprechenden Spielernamen der Spieler enthält, die dem Spiel beigetreten sind.

Nachdem die MultisweeperClients diese Daten erhalten haben, führen Sie das Get-Game-World-Protokoll aus.

8.2.4. Join-Running-Game-Protokoll

Dieses Protokoll wird ausgeführt, wenn ein Spieler einem bereits laufenden Spiel beitritt.

Nachdem ein MultisweeperClient mit dem Game-Discovery-Protokoll die auf Clippee verfügbaren Spiele in Erfahrung gebracht hat, entscheidet sich der Benutzer für eines dieser Spiele, um an diesem teilzunehmen. Das Spiel wird durch seine `gameID` identifiziert. Er muss die Netzwerkadresse eines der bereits beteiligten MultisweeperClients in Erfahrung bringen, um danach das aktuelle Spielbrett herunterzuladen.

MultisweeperClient ⇨ GamePeer

`PlayGameMessage(String playerIdStr, long playerID, int gameID, int listeningPort)`

Parameter:

`playerIdStr`	Spielername
`PlayerID`	eindeutige Player-Id
`GameID`	eindeutige Game-Id des Spiels, dem der Client beitreten möchte
`listeningPort`	Der Port, auf dem der neue Client auf eingehende Verbindungen hört, wenn später andere Clients seinem Spiel beitreten möchten.

Der GamePeer fügt den MultisweeperClient zum Spiel hinzu. Es gibt nun keine Wartezeit. Das Spiel ist ja schon am Laufen. Dem Spieler wird direkt eine `MultisweeperGameReadyMessage` zugeschickt.

GamePeer ⇨ MultisweeperClient

`MultisweeperGameReadyMessage(String ip, int port, int gameId, Vector players)`

Parameter:

`ip`	siehe oben
`port`	siehe oben
`gameId`	siehe oben
`players`	siehe oben

Nachdem der Joining-MultisweeperClient diese Daten erhalten hat, führt er das GetGame-World-Protokoll aus.

8.2.5. Get-Game-World-Protokoll

Am Ende des Join-Waiting-Game- bzw. Join-Running-Game-Protokolls erhalten alle beteiligten MultisweeperClients mit der `MultisweeperGameReadyMessage` eine IP und einen Port. Dies sind die Adressdaten des MultisweeperClients, von dem das gültige Spielbrett heruntergeladen werden soll. Dieser MultisweeperClient wird `King` genannt.

MultisweeperClient ⇨ King
`GetGameWorldMessage()`

Der `King` erstellt ein `GameWorldObjekt` aus seinem eigenen Spielbrett und sendet diese Daten an den MultisweeperClient.

King ⇨ MultisweeperClient
`GameWorldMessage(GameWorldObject myGameWorld)`

Das `GameWorldObject` wird an den GUI-Layer übergeben. Dieser sorgt dann für die korrekte Darstellung der erhaltenen GameWorld. Das Spiel ist nun bereit.

8.2.6. Nachrichten während des Spielverlaufs

Aufdecken eines freien Feldes
Die `UncoverMessage` wird von einem MultisweeperClient an alle anderen MultisweeperClients versendet, sobald der Spieler auf ein freies Feld klickt. Das Feld wird bei seinen Mitspielern nach dem Empfang der `UncoverMessage` auf dem Spielbrett auch aufgedeckt.

MultisweeperClient ⇨ alle MultisweeperClients des Spiels
`UncoverMessage(long playerIdInt, int x, int y)`

Parameter:

`playerIdInt`	Spieler-Id des Spielers, der das Feld aufgedeckt hat.
`x, y`	Koordinaten des Feldes, das aufgedeckt wurde.

Setzen/Entfernen eines Flags
Beim Setzen eines Flags wird eine `SetFlagMessage` an alle anderen MultisweeperClients des Spiels versendet. Das Feld wird beim Empfang der Nachricht bei den Mitspielern als Remoteflag auf dem Spielbrett gesetzt bzw. entfernt.

MultisweeperClient ⇨ alle MultisweeperClients des Spiels
`SetFlagMessage(int index, int x, int y, long playerId)`

Parameter:

`index`	`-1`: Flag wurde entfernt, `1`: Flag wurde gesetzt
`x, y`	Koordinaten des Feldes, wo das Flag gesetzt wurd
`playerId`	Spieler-Id des Spielers, der das Flag gesetzt/entfernt hat
`playerId`	Spieler-Id des Spielers, der das Feld aufgedeckt hat

Verlassen des Spiels: Das Leave-Protokoll
Wenn ein Spieler das Spiel vorzeitig verlässt, so sendet er eine `ExitGameMessage` an alle Mitspieler. Diese entfernen den Spieler dann aus dem P2P-Netzwerk und vom Spielbrett. Es wird zusätzlich eine `ExitGameMessage` an den GamePeer gesendet, damit der das Spiel verlassende Spieler aus dem Spiel entfernt werden kann.

MultisweeperClient ⇨ alle MultisweeperClients des Spiels
`ExitGameMessage(long playerId, int gameId)`

Parameter:

`playerId`	siehe oben
`gameId`	siehe oben

MultisweeperClient ⇨ GamePeer
`ExitGameMessage(long playerId, int gameId)`

Parameter:

`playerId`	siehe oben
`gameId`	siehe oben

Klicken auf eine Mine
Wenn ein Spieler auf eine Mine klickt, so werden seine Punkte zurückgesetzt. Damit die Mitspieler die Punkte für den betreffenden Spieler auch zurücksetzen, versendet der Spieler, der auf eine Mine geklickt hat, eine `ResetPointsMessage` an seine Mitspieler.

MultisweeperClient ⇨ alle MultisweeperClients des Spiels
`ResetPointsMessage(long playerId)`

Parameter:

`playerId`	siehe oben

8.2.7. Entdecken von Clients, die nicht mehr antworten

Dieses Protokoll stellt sicher, dass Clients, die nicht mehr antworten, erkannt werden, damit sie aus dem P2P-Netzwerk und aus dem Spiel entfernt werden können. Dies ist wichtig, da es sonst Spiele geben kann, die ewig existieren, da vermeintlich noch Spieler in einem Spiel vorhanden sind. In Wirklichkeit haben diese Spieler das Spiel längst verlassen. Einerseits muss der MultisweeperPeer, der ein Spiel verwaltet, ständig überprüfen, ob noch alle beteiligten Spieler antworten. Andererseits muss auch jeder MultisweeperClient ständig überprüfen, ob all seine Mitspieler noch antworten. Das folgende einfache Protokoll wird periodisch durchgeführt. Antwortet ein Peer nicht mehr, so wird er aus dem Spiel und aus dem P2P-Netzwerk entfernt.

GamePeer ⇨ MultisweeperClient
`PingMessage()`

MultisweeperClient ⇨GamePeer
`PingReplyMessage()`

MultisweeperClient1 ⇨ MultisweeperClient2
`PingMessage()`

MultisweeperClient2 ⇨ MultisweeperClient1
`PingReplyMessage()`

9. Probleme, die sich ergeben haben

Ich werde kurz einige Probleme aufzeigen, die sich bei der Arbeit ergeben haben.

Da das Spiel verteilt ist, kann es vorkommen, dass zwei Spieler gleichzeitig auf dasselbe Feld klicken. Was soll in diesem Fall geschehen? Ich bin zu folgender Lösung gekommen:

Wenn ein Spieler lokal einen Punkt erreicht hat, so bekommt er ihn auch. Das heisst, zwei Spieler können dasselbe Feld aufdecken und erhalten beide einen Punkt gutgeschrieben. Bei der Entwicklung der Protokolle wurde davon ausgegangen, dass es keine böswilligen Spieler gibt, die das Protokoll manipulieren.

Eines der grössten Probleme waren erwartungsgemäss die Inkonsistenzen. Da es keinen zentralen Server gibt, muss immer sichergestellt werden, dass alle Spieler eine konsistente Sicht auf das Spielfeld haben.

Wenn ein Spieler das Spiel verlassen hat, kam es oft vor, dass der Spieler nicht richtig aus Clippee und aus dem MultisweeperGame entfernt wurde. Ich habe daraufhin das Leave-Protokoll eingeführt, damit sich alle sauber abmelden. Es kam aber immer noch vor, dass Spieler nach dem mutwilligen beenden des Java-Prozesses in den Spielen verblieben, weil in diesem Fall das Leave-Protokoll nicht ausgeführt wird. Dazu habe ich einen Connection-Listener programmiert, der dann ausgeführt wird, wenn eine Verbindung plötzlich unterbrochen wird. Zusätzlich habe ich das Ping-Protokoll eingeführt, dass auf einer Verbindung periodisch überprüft, ob das Gegenüber noch antwortet. Ist dies nicht mehr der Fall, so wird die Verbindung abgebaut, und der Spieler wird aus dem Spiel entfernt.

Auch bei den Protokollen war es zum Teil schwer, diese so zu konstruieren, dass sie in jedem Fall korrekt ablaufen. Das Join-Game-Protokoll funktioniert noch nicht in jedem Fall korrekt. Wenn der King, von dem die anderen Spieler die Konfiguration des Spielbrettes herunterladen, ausgeschaltet wird, bevor alle Mitspieler die GameWorld heruntergeladen haben, so kann das Spiel nicht korrekt gestartet werden. Um dies zu beheben wäre der Aufwand zu gross gewesen. Ein Ansatz wäre, dass nach einem nicht erfolgreichen Herunterladen der GameWorld bei anderen Spielern nach der GameWorld gesucht wird.

Einige Probleme sind mit dem Best-Effort-Ansatz gelöst worden. Es ist bestimmt noch möglich, dass in einer ausgefallenen Konfiguration, besonders wenn viele Spieler beteiligt sind, Inkonsistenzen auftreten können. Ich habe das Spiel aber ausgetestet und alle Fehler beseitigt, die ein problemloses Spielen unter normalen Bedingungen beeinträchtigten.

10. Eigene Erfahrungen

Ich habe bei der Arbeit einiges gelernt. Zum einen musste ich mich in die bestehende Software Clippee einarbeiten, Code lesen und die enthaltenen Konzepte verstehen. An der ETH muss man oft nur kleine 10-Zeilen-Prograämmchen verstehen. Ich habe einige Zeit aufgewendet, um mich durch den bestehenden Code zu lesen und die Zusammenhänge zu verstehen, bevor ich mit der eigentlichen Arbeit beginnen konnte.

Zum anderen war es eine besondere Herausforderung ein komplexes verteiltes Programm zu erstellen. Es hat mich zum Teil ganzes Kopfzerbrechen gekostet, bei 20 gleichzeitig laufenden Threads den Überblick zu behalten, während die Threads untereinander Nachrichten austauschten. Die tückischen Fehler sind manchmal erst bei 10 Spielern aufgetreten, weil erst dann eine bestimmte Fehlersituation eingetreten ist. Einige der Fehler waren auch nicht deterministisch und konnten nur schwer reproduziert werden oder sind erst dann wieder aufgetreten wenn ich ihre Existenz bereits vergessen hatte.

Das Debuggen von verteilten Programmen war teils eine spannende aber nicht selten auch nervenaufreibende Angelegenheit. Im Fehlerfall konnte ich mich selten auf den Compiler verlassen. Es war oft schwer herauszufinden, wo der eigentliche Fehler liegt. War er beim Erstellen, beim Serialisieren, beim Versenden, beim Empfangen oder beim Deserialisieren? der Nachricht. Da war oft einfach Ausprobieren angesagt. Ich konnte aber mit der Zeit etwas systematischer vorgehen, da ich die häufigsten Fehlerquellen bereits kennengelernt hatte.

Da ich das gut funktionierende Kommunikationssystem von Clippee zur Verfügung hatte, konnte ich auch einige einfache Protokolle entwerfen. Dies war insofern spannend, als ich noch nie selbst ein Protokoll entworfen habe.

Es hat mir Spass gemacht das spannende Multiplayer-Game zu entwerfen und zu implementieren. Ich hoffe, dass die Spieler mit Multisweeper auch Spass haben werden!