Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. Peer Kröger
Dr. Daniel Kaltenthaler, Dr. Johannes Lohrer

München, 12.06.2018

Softwareentwicklungspraktikum
SS 2018 – Robo Rally

Protokoll Version 1.0

Neue Funktionen

In dieser Version neu implementiert sind folgende Punkte:

- Erweiterung des Spielfeldes: alternative Spielfelder, neue Syntax für Spielfelderstellung
- Erweiterung um zehn Upgradekarten
- Passive Upgrades: Admin Privilege, Rear Laser, Deflector Shield, Cache Memory, Memory Stick
- Temporäre Upgrades: Boink, Hack, Memory Swap, Spam Blocker, Repeat Routine

1 Grundlagen

Die Daten werden über TCP als JSON-Objekte übertragen. Als Zeichensatz wird hierfür UTF-8 verwendet, wobei unterschiedliche Nachrichtentypen durch sogenannte Wrapper-Objekte modelliert sind.

Ein Objekt darf dabei stets nur eine Nachricht enthalten und sieht folgendermaßen aus:

```
{
    "messageType" : "sample",
    "messageBody" : {
        "keyOne" : valueOne,
        "keyTwo" : valueTwo
    }
}
```

Statt eines primitiven "value" können hier jedoch auch komplexere JSON-Objekte auftreten. So kann ein "messageBody" z.B. mehrere Objekte und / oder Listen enthalten. Auch Kombinationen aus primitiven Typen und Objekten sind denkbar.

2 Verbindungsaufbau

Verbindung erfolgreich
Nach dem Aufbau der TCP-Verbindung sendet der Server an den Client seine Protokollversion.

```
{
    "messageType" : "HelloClient",
    "messageBody" : {
        "protocol" : 0.1
    }
}
```

Gruppenidentifikation
Anschließend antwortet der Client mit Informationen zu seiner benutzten Protokollversion, dem Gruppennamen und ob es sich um den Login einer KI handelt.

```
{
    "messageType" : "HelloServer",
    "messageBody" : {
        "protocol" : 0.1,
        "group" : "TolleTrolle",
        "isAI" : false
    }
}
```

Sollte der Server die Protokollversion der Gruppe nicht unterstützen, so ist die Verbindung abzubrechen. Genaueres hierzu im Teil: **Besondere Nachrichten**

Übergabe der Spieler ID
Ist alles in Ordnung, erhält der Client vom Server eine Spielernummer zugewiesen.
Diese sollte im positiven Bereich liegen und muss natürlich einzigartig sein.

```
{
    "messageType" : "Welcome",
    "messageBody" : {
        "id" : 9001
    }
}
```

3 Lobby

Setzen des Spielernamens und der Figur
Nach dem erfolgreichen Verbindungsaufbau kann ein Spieler seinen Namen und seine Spielfigur auswählen. Beide sollten einzigartig sein, werden aber in einer Nachricht verschickt.

Bedenken Sie ggf. die Nutzerfreundlichkeit, wenn eine der beiden Werte schon vergeben ist!

```
{
    "messageType" : "PlayerValues",
    "messageBody" : {
        "name" : "Nr. 5",
        "figure" : 2
    }
}
```

Bestätigung der Spielerauswahl
Ein bereits vergebener Name oder eine vergebene Figur sollen, wie im Bereich **Besondere Nachrichten** beschrieben, quittiert werden. Valide Wünsche werden für alle verbundenen Clients sichtbar bestätigt.

```
{
    "messageType" : "PlayerAdded",
    "messageBody" : {
        "player" : {
            "id" : 9001,
            "name" : "Nr. 5",
            "figure" : 2
        }
    }
}
```

Bedenken Sie, dass neu hinzukommende Clients auch nachträglich über bereits vorhandene Spieler informiert werden müssen.

Bereitschaft signalisieren
Sobald ein Spieler seine Auswahl erfolgreich getroffen hat, kann er dem Server signalisieren bereit zu sein. Diese Meinung kann er via Boolean aber auch zurückziehen!

```
{
    "messageType" : "SetStatus",
    "messageBody" : {
        "ready" : "true"
    }
}
```

Bereitschaft bestätigen oder widerrufen
Von Seiten des Servers wird der Spielerstatus an alle verbundenen Clients verteilt.

```
{
    "messageType" : "PlayerStatus",
    "messageBody" : {
        "id" : 9001,
        "ready" : true
    }
}
```

Auch diesen Status müssen neu verbindende Clients empfangen.

Sind alle Spieler bereit (mind. 2!), erstellt der Server die Karte und teilt diese den Anderen mit.

Jedes Feldobjekt beinhaltet die einzelnen Feldtypen und deren Orientierung.

- (Neu!) Die Positionen spiegeln sich in einem Map-Array[][] wieder.
- (Neu!) Valide Feldtypen sind: Empty, StartPoint, Belt, RotatingBelt, PushPanel, Gear, Pit, EnergySpace, Wall, Laser, Antenna, ControlPoint
- Als Grundlage für die Orientierung dient die Aktion der Felder: Bei einem Belt die Laufrichtung, bei einem Push-Panel die Pushrichtung, ect.
- Die Orientierung von "Gear" ist als Drehrichtung zu interpretieren. Hierbei ist z.B. im Uhrzeigersinn als "right" anzugeben.
- "RotatingBelt" hat zwei Orientierungen. Die erste gibt die normale Laufrichtung an, die zweite die Richtung, aus der die Drehung erfolgt. Zusätzlich gibt es einen Boolean, um zwischen Kreuzung (true) und Kurve (false) zu unterscheiden.
- "EnergySpaces", "Laser" und "ControlPoint" haben einen zusätzlichen Wert "count" für die vorhandene Energie und die Anzahl der Laser bzw. Nummer des Punktes.
- "Belt" hat einen zusätzlichen Wert für Speed (1 / 2 - Grün / Blau).
- "PushPanel" inkludiert eine Liste der Register, in denen sie aktiv werden.
- "Wall" kann mehrere Orientierungen haben. Diese geben an, welche Richtungen geblockt werden.
- (Neu!) Die Startzone wird nun mit übertragen. Es ist jetzt möglich, auch erweiterte Spielfelder zu nutzen.

Beispiel für eine Map mit komplexen Feldern

Nach oben verlaufender (blauer) Belt, der nach links rotiert und in eine Kreuzung mündet.
Panel, das nach links schiebt, wenn Register zwei oder vier aktiv sind.
Eine Wall, die oben und links das Feld begrenzt und einen Laser nach unten verschießt.


```
{
    "messageType" : "GameStarted",
    "messageBody" : {
        "mapY" : [{
                "mapX" : [{
                        "field" : [{
                                "type" : "RotatingBelt",
                                "speed" : 2,
                                "isCrossing" : true,
                                "orientations" : [ "up", "left" ]
                            }
                        ]
                    },{
                        "field" : [{
                                "type" : "PushPanel",
                                "orientations" : [ "left" ],
                                "registers" : [ 2, 4 ]
                            }
                        ]
                    }
                ]
            },{
                "mapX" : [{
                        "field" : [{
                                "type" : "Wall",
                                "orientations" : [ "up", "left" ]
                            },{
                                "type" : "Laser",
                                "orientations" : [ "down" ],
                                "count" : 1
                            }
                        ]
                    }
                ]
            }
        ]
    }
}
```

4 Chatnachrichten

Privat senden
Spieler sollen untereinander Nachrichten austauschen können. Diese werden vom Server verteilt. Ein Client kann eine private Nachricht an Spieler ID 9001 wie folgt veranlassen:

```
{
    "messageType" : "SendChat",
    "messageBody" : {
        "message" : "Yoh, Bob! How is your head doing after last Night?",
        "to" : 9001
    }
}
```

Privat verteilen
Der Client steht in der Pflicht, diese Nachricht nur dem richtigen Spieler anzuzeigen.

```
{
    "messageType" : "ReceivedChat",
    "messageBody" : {
        "message" : "Yoh, Bob! How is your head doing after last Night?",
        "from" : "Gustav Gans",
        "private" : true
    }
}
```

Öffentlich senden
Im Falle einer Nachricht für alle Spieler wird "to" zu -1.

```
{
    "messageType" : "SendChat",
    "messageBody" : {
        "message" : "Hi all! I will crush your robots in no time!",
        "to" : -1
    }
}
```

Öffentlich verteilen
Hier kann die Nachricht normal im Chatfenster ausgegeben werden.

```
{
    "messageType" : "ReceivedChat",
    "messageBody" : {
        "message" : "Oh my, this was some nasty stuff. I cencored it!",
        "from" : "Gustav Gans",
        "private" : false
    }
}
```

5 Besondere Nachrichten

Fehlerhafte Nachrichten
Bei einem Übertragungsfehler soll der Server den Client entsprechend informieren. Dies geschieht, je nach Servereinstellung, mit detaillierter Beschreibung oder nur als rudimentärer Hinweis.
Im Allgemeinen verlangt ein Übertragungsfehler nach einem erneuten Senden der fehlerhaften Mitteilung. Bedenken Sie dies bei Ihrer Implementation.

```
{
    "messageType" : "Error",
    "messageBody" : {
        "error" : "Ups! That did not work. Try to adjust something."
    }
}
```

Verbindungsverlust
Es kann jederzeit vorkommen, dass ein Client die Verbindung verliert.
Dies soll den Anderen natürlich mitgeteilt werden. Wie Sie mit dieser Situation dann umgehen, ist Ihnen überlassen.

Auch eine erneute Verbindung kann gestattet werden. Natürlich macht dies nur Sinn, wenn der Roboter vorher nicht entfernt wurde.

Mögliche Optionen sind also

- Remove: Entfernt den Roboter des Clients aus dem Spiel.
- AIControll: Eine KI übernimmt die Kontrolle.
- Ignore: Der Client wird ignoriert, verbleibt aber (regungslos) im Spiel. Sein Roboter kann weiterhin Interaktionen auslösen, wird aber nicht mehr aktiv bewegt.
- Reconnect: Der Client ist wieder online und verbunden.

```
{
    "messageType" : "ConnectionUpdate",
    "messageBody" : {
        "id" : 2,
        "connected" : false,
        "action" : "AIControll"
    }
}
```

Es ist nicht nötig, jede dieser Optionen umzusetzen. Es ist ausreichend, die "Remove" Version zu implementieren.

6 Spielkarten

Karten spielen
Basiskarten werden ganz rudimentär als String übertragen. Natürlich sollten diese Karten in Ihrem Model trotzdem in Objektform hinterlegt sein!

```
{
    "messageType" : "PlayCard",
    "messageBody" : {
        "card" : "MoveI"
    }
}
```

Der Server gibt dies an die anderen Clients weiter.

```
{
    "messageType" : "CardPlayed",
    "messageBody" : {
        "playerID" : 1,
        "card" : "MoveI"
    }
}
```

Programmierkarten
MoveI, MoveII, MoveIII, TurnLeft, TurnRight, UTurn, BackUp, PowerUp und Again.

Schadenskarten
Spam, Wurm, Virus und Trojaner.

Besondere Programmierkarten
Energy, Sandbox, Weasel, Speed, Spam und Repeat.

Upgradekarten
Da im Rahmen des SEP nur zehn der Karten eingebunden werden müssen, wird jede einzelne mehrfach vorkommen können. Sollten Sie mehr als die hier vorgestellten Karten einbinden (freiwillig), passen Sie bitte die Multiplikatoren entsprechend an.

Passiv, je 5x: AdminPrivilege, RearLaser, DeflectorShield, CacheMemory und MemoryStick.
Temporär, je 3x: Boink, Hack, MemorySwap, SpamBlocker und RepeatRoutine.

7 Spielzug abhandeln

Aktiven Spieler setzen
Der Server gibt eine kurze Nachricht aus, um den aktuell aktiven Spieler zu bestimmen. Entgegen der offiziellen Anleitung beginnt derjenige, der zuerst die Verbindung aufgebaut hat.

```
{
    "messageType" : "CurrentPlayer",
    "messageBody" : {
        "playerID" : 3
    }
}
```

Aktive Spielphase setzen
Die aktuelle Phase wird mittels einer eigenen ID übertragen. Hierbei gilt:
0 => Aufbauphase, 1 => Upgradephase, 2 => Programmierphase, 3 => Aktivierungsphase

```
{
    "messageType" : "ActivePhase",
    "messageBody" : {
        "phase" : 3
    }
}
```

7.1 Aufbauphase

Startpunkt setzen
Spieler setzen ihren Startpunkt wie folgt:

```
{
    "messageType" : "SetStartingPoint",
    "messageBody" : {
        "position" : [ 7, 4 ]
    }
}
```

Bestätigen des Startpunktes
Wenn die gewünschte Position valide ist, werden alle Spieler darüber benachrichtigt.

```
{
    "messageType" : "StartingPointTaken",
    "messageBody" : {
        "playerID" : 1,
        "position" : [ 7, 4 ]
    }
}
```

7.2 Upgradephase

Shop auffüllen
Ist der Shop nicht voll, wird den Clients mitgeteilt, welche neuen Upgrades zur Verfügung stehen. In der ersten Runde wird diese Nachricht im Rahmen der Aufbauphase das erste Mal verschickt.

```
{
    "messageType" : "RefillShop",
    "messageBody" : {
        "cards" : [ // 1-4 cards ]
    }
}
```

Shop erneuern
Wurde in der letzten Runde kein Upgrade gekauft, wird der Shop komplett erneuert.

```
{
    "messageType" : "ExchangeShop",
    "messageBody" : {
        "cards" : [ // 4 cards ]
    }
}
```

Upgrades kaufen
Der aktive Spieler kann eine Karte kaufen, oder ablehnen. Diese Nachricht muss in jedem Fall verschickt werden.

```
{
    "messageType" : "BuyUpgrade",
    "messageBody" : {
        "buying" : true,
        "card" : // card if true, empty if false
    }
}
```

Kaufbestätigung
Ist beim Kaufprozess alles in Ordnung (Spieler an der Reihe, genug Energie), bestätigt der Server dies für alle Spieler sichtbar. Die Energiekosten werden behandelt, wie im Abschnitt Aktionen beschrieben.

```
{
    "messageType" : "UpgradeBought",
    "messageBody" : {
        "playerID" : 1,
        "card" : "Boink"
    }
}
```

7.3 Programmierphase

Spieler zieht Karten
Es werden die obersten neun Programmierkarten gezogen. Die Nachricht vom Server sollte natürlich nur von dem jeweils betroffenen Spieler empfangen werden können!

```
{
    "messageType" : "YourCards",
    "messageBody" : {
        "cards" : [ //cards ]
    }
}
```

Die anderen Spieler werden lediglich über die Anzahl der Karten benachrichtigt.

```
{
    "messageType" : "NotYourCards",
    "messageBody" : {
        "playerID" : 1,
        "cards" : 9
    }
}
```

Programmierkarten mischen
Sind nicht mehr genug Karten auf dem Kartenstapel, muss der Ablegestapel gemischt werden.

```
{
    "messageType" : "ShuffleCoding",
    "messageBody" : {
        "playerID" : 1
    }
}
```

Karten auswählen
Die Kartenauswahl wird vom Spieler zum Server übertragen, jede Karte einzeln. Überschreibungen und ein leeren des Registers sind möglich.

```
{
    "messageType" : "SelectCard",
    "messageBody" : {
        "card" : "again" //can also be empty to clear a register,
        "register" : 5
    }
}
```

Kartenwahl bestätigen
Der Server gibt das belegte Register, natürlich ohne Karteninformation, an alle weiter.

```
{
    "messageType" : "CardSelected" ,
    "messageBody" : {
        "playerID" : 2,
        "register" : 5
    }
}
```

Auswahl beendet
Sobald ein Spieler die fünfte Karte gelegt hat, sind keine Änderungen mehr möglich! Dies wird für alle sichtbar übertragen.

```
{
    "messageType" : "SelectionFinished" ,
    "messageBody" : {
        "playerID" : 1
    }
}
```

Timer gestartet
Als Folge des ersten fertigen Spielers startet der 30 Sekunden Timer.

```
{
    "messageType" : "TimerStarted" ,
    "messageBody" : {}
}
```

Timer beendet
Zum Ende des Timers wird eine Meldung an die Clients geschickt. Diese beinhaltet auch die Information über evtl. zu langsam reagierende Spieler.

```
{
    "messageType" : "TimerEnded" ,
    "messageBody" : {
        "playerIDs" : [ 2, 3, 4 ]
    }
}
```

Hand abwerfen und blind ziehen
Ist ein Spieler dem Timer zum Opfer gefallen, muss er alle Karten abwerfen, bevor der Server (regelkonform) alle fünf Register auffüllt. Das Ergebnis wird dem Spieler dann mitgeteilt.

```
{
    "messageType" : "DiscardHand",
    "messageBody" : {
        "playerID" : 2
    }
}

{
    "messageType" : "CardsYouGotNow",
    "messageBody" : {
        "cards" : [ //all the selected register cards ]
    }
}
```

7.4 Aktivierungsphase

Aktive Programmierkarten
Jede der fünf Runden wird durch eine Nachricht mit den nun aktiven Karten eingeläutet. Es wird jeweils nur die Karte aus dem aktuellen Register angezeigt.

```
{
    "messageType" : "CurrentCards",
    "messageBody" : {
        "activeCards" : [{
                "playerID" : 1,
                "card" : //card
            },{
                "playerID" : 2,
                "card" : //card
            }
        ]
    }
}
```

Programmierkarte spielen
Um auf potentielle Upgradekarten reagieren zu können, ist es nötig, jede gespielte Karte vom jeweiligen Spieler absegnen zu lassen. Zur Vereinfachung wird folgender Aufruf unterstützt:

```
{
    "messageType" : "PlayIt",
    "messageBody" : {}
}
```

Hinweis: Der "playCard" Aufruf soll in dieser Phase ausschließlich für Upgradekarten funktionieren.

8 Aktionen, Ereignisse und Effekte

Bewegung
Bewegungen einer Spielfigur werden vom Server übertragen, jedoch ohne den Auslöser zu nennen. Wichtig: Es handelt sich rein um die Bewegung zwischen Feldern. Ohne Drehung!

```
{
    "messageType" : "Movement",
    "messageBody" : {
        "playerID" : 1,
        "to" : [2, 5]
    }
}
```

Schadensmeldungen
Es werden immer alle auf einmal übertragen. Wenn ein Spieler also drei Schaden bekommt, wird trotzdem nur eine Nachricht verschickt.

```
{
    "messageType" : "DrawDamage",
    "messageBody" : {
        "playerID" : 1,
        "cards" : [ //Schadenskarten ]
    }
}
```

Schießende Roboter
Für Animationen hilfreich, ansonsten redundant.

```
{
    "messageType" : "PlayerShooting",
    "messageBody" : {}
}
```

Neustart
Ein Neustart benötigt keine genaueren Details.

```
{
    "messageType" : "Reboot",
    "messageBody" : {
        "playerID" : 1
    }
}
```

Drehung
Die Richtung wird hier mit "clockwise", bzw. "counterClockwise" angegeben.

```
{
    "messageType" : "PlayerTurning",
    "messageBody" : {
        "playerID" : 1,
        "direction" : "clockwise"
    }
}
```

Energie
Keine besondere Logik nötig an dieser Stelle. Die Energie wird einfach als Hinweis übertragen.

```
{
    "messageType" : "Energy",
    "messageBody" : {
        "playerID" : 1,
        "count" : 1
    }
}
```

Erreichte Checkpoints
Hier entspricht der Key "number" den Kontrollpunktmarken der physischen Version. Ein Spieler mit dem Wert 3 hat also die ersten drei Checkpoints erreicht.

```
{
    "messageType" : "CheckpointReached",
    "messageBody" : {
        "playerID" : 1,
        "number" : 3
    }
}
```

Spielsieg
Sobald ein Spieler den letzten Checkpoint erreicht, gilt das Spiel als gewonnen.

```
{
    "messageType" : "GameWon",
    "messageBody" : {
        "playerID" : 1
    }
}
```