Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. Peer Kröger
Maximilian Hünemörder, Ludwig Zellner

München, 02.07.2019

Softwareentwicklungspraktikum
SS 2019 – Robo Rally

Protokoll Version 2.0

Übersicht

- (Neu) Support für Upgradekarten
- (Offen) nichts

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": "keyTwo"
  }
}
```

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 seine Protokollversion an den Client.

```
{
  "messageType": "HelloClient",
  "messageBody": {
    "protocol": "Version 2.0"
  }
}
```

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": {
    "group": "TolleTrolle",
    "isAI": false,
    "protocol": "Version 2.0"
  }
}
```

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": {
    "playerID": 42
  }
}
```

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.

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

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": {
    "playerID": 42,
    "name": "Nr. 5",
    "figure": 5
  }
}
```

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": {
    "playerID": 42,
    "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.

- Die Karte wird als $\text{List}_X \langle \text{List}_Y \langle \text{List}_{\text{Field}} \langle \text{Field}_{\text{Typ}} \rangle \rangle \rangle$ übertragen und ist wie ein Koordinatensystem aufgebaut. x0,y0 ist also link-unten. x5,y0 ist rechts vom Startpunkt.
- Felder können mehr als einen Typus haben. Daher die dritte verschachtelte Liste, um Felder wie "eine Wand mit Laser" zu modellieren.
- Leere Felder (z.B. bei nicht rechteckigen Spielfeldern) werden mit **null** belegt.
- (Neu!) Valide Feldtypen sind: Empty, StartPoint, Belt, RotatingBelt, PushPanel, Gear, Pit, EnergySpace, Wall, Laser, Antenna, CheckPoint, RestartPoint
- Als Grundlage für die Orientierung dient die Aktion der Felder: Bei einem Belt die Laufrichtung, bei einem Push-Panel die Pushrichtung, usw.
- "RotatingBelt" hat zwei Orientierungen. Die erste gibt die Laufrichtung vor der Drehung an, die zweite die finale Laufrichtung. Zusätzlich gibt es einen Boolean, um zwischen Kreuzung (true) und Kurve (false) zu unterscheiden.
- "Belt" hat einen zusätzlichen Wert für Speed (1 / 2 - Grün / Blau).
- Die Orientierung von "Gear" ist als Drehrichtung zu interpretieren. Hierbei ist sie z.B. im Uhrzeigersinn als "right" anzugeben.
- Die "Antenna" Orientierung gibt die Senderichtung an.
- "EnergySpaces", "Laser" und "CheckPoint" haben einen zusätzlichen Wert "count" für die vorhandene Energie und die Anzahl der Laser bzw. Nummer des Punktes.
- "PushPanel" inkludiert eine Liste der Register, in denen sie aktiv werden.
- "Wall" kann mehrere Orientierungen haben. Diese geben die Seiten an, die mit einer Wand geblockt werden.
- Ab jetzt sollen Sie alle Spielfelder unterstützen, die im Regelwerk hinterlegt sind.

Beispiel für eine Map mit komplexen Feldern

x0;y0: Nach oben verlaufender (blauer) Belt, der nach links rotiert und in eine Kreuzung mündet.
x0;y1: Panel, das nach links schiebt, wenn Register zwei oder vier aktiv sind.
x1;y0: Wall, die oben und links begrenzt und einen Doppel-Laser nach unten verschießt.
x1;y1: Leeres Feld


```
{
  "messageType": "GameStarted",
  "messageBody": {
    "gameMap": [
      [
        [
          {
            "type": "RotatingBelt",
            "speed": 2,
            "isCrossing": true,
            "orientations": [
              "up",
              "left"
            ]
          }
        ],
        [
          {
            "type": "PushPanel",
            "orientations": [
              "left"
            ],
            "registers": [
              2
            ]
          }
        ]
      ],
      [
        [
          {
            "type": "Wall",
            "orientations": [
              "up",
              "right"
            ]
          },
          {
            "type": "Laser",
            "orientations": [
              "down"
            ],
            "count": 2
          }
        ],
        [
          null
        ]
      ]
    ]
  }
}
```

4 Chatnachrichten

Privat senden

Spieler sollen untereinander Nachrichten austauschen können. Diese werden vom Server verteilt. Ein Client kann eine private Nachricht an Spieler ID 42 wie folgt veranlassen:

```
{
  "messageType": "SendChat",
  "messageBody": {
    "message": "Yoh, Bob! How is your head doing after last night?",
    "to": 42
  }
}
```

Privat verteilen

Der Server steht in der Pflicht, diese Nachricht nur dem richtigen Spieler zu schicken.

```
{
  "messageType": "ReceivedChat",
  "messageBody": {
    "message": "Yoh, Bob! How is your head doing after last night?",
    "from": 42,
    "isPrivate": 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": "Hi all! I will crush your robots in no time.",
    "from": 42,
    "isPrivate": false
  }
}
```

5 Besondere Nachrichten

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

```
{
  "messageType": "Error",
  "messageBody": {
    "error": "Whoops. 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": {
    "playerID": 9001,
    "isConnected": false,
    "action": "AIControl"
  }
}
```

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 Modell trotzdem in Objektform hinterlegt sein!

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

Der Server gibt dies an die anderen Clients weiter.

```
{
  "messageType": "CardPlayed",
  "messageBody": {
    "playerID": 2,
    "card": "MoveI"
  }
}
```

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

Wichtig: "PlayCard" wird auch bei Revolten verwendet. TurnLeft, TurnRight und UTurn sind hier valide und müssen sich nicht im Besitz befinden. Die Server sollen diesen Sonderfall korrekt behandeln.

Schadenskarten
Spam, Worm, Virus und Trojan.

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

Upgradekarten
Im Rahmen des SEP müssen nur zehn der Karten eingebunden werden. Sie können zwar mehr als die hier vorgestellten Karten umsetzen, bedenken Sie aber, dass nicht jeder Server dies unterstützen wird!

Passiv: AdminPrivilege, RearLaser, DeflectorShield, CacheMemory und MemoryStick.
Temporär: 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": 7
  }
}
```

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 wählen
Spieler setzen ihren Startpunkt wie folgt:

```
{
  "messageType": "SetStartingPoint",
  "messageBody": {
    "x": 4,
    "y": 2
  }
}
```

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

```
{
  "messageType": "StartingPointTaken",
  "messageBody": {
    "x": 4,
    "y": 2,
    "playerID": 42
  }
}
```

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": [
      "card1",
      "card2",
      "...",
      "card6"
    ]
  }
}
```

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

```
{
  "messageType": "ExchangeShop",
  "messageBody": {
    "cards": [
      "card1",
      "card2",
      "...",
      "card6"
    ]
  }
}
```

Upgrades kaufen
Der aktive Spieler kann eine Karte kaufen, oder ablehnen. Diese Nachricht muss in jedem Fall verschickt werden. Der Wert für "card" ist dann ggf. Null.

```
{
  "messageType": "BuyUpgrade",
  "messageBody": {
    "isBuying": true,
    "card": "Boink"
  }
}
```

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": 42,
    "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": {
    "cardsInHand": [
      "card1",
      "..."
    ],
    "cardsInPile": 9001
  }
}
```

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

```
{
  "messageType": "NotYourCards",
  "messageBody": {
    "playerID": 42,
    "cardsInHand": 9,
    "cardsInPile": 9001
  }
}
```

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

```
{
  "messageType": "ShuffleCoding",
  "messageBody": {
    "playerID": 42
  }
}
```

Karten auswählen
Die Kartenauswahl wird vom Spieler zum Server übertragen, jede Karte einzeln. Überschreibungen und ein Leeren des Registers sind möglich, indem "card = Null" gesetzt wird.

```
{
  "messageType": "SelectedCard",
  "messageBody": {
    "card": "Again",
    "register": 5
  }
}
```

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

```
{
  "messageType": "CardSelected",
  "messageBody": {
    "playerID": 42,
    "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": 42
  }
}
```

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": [
      1,
      3,
      6
    ]
  }
}
```

Hand abwerfen und blind ziehen
Diese Karten werden in der vorgegebenen Reihenfolge auf die noch offenen Register gelegt.

```
{
  "messageType": "CardsYouGotNow",
  "messageBody": {
    "cards": [
      "card1",
      "..."
    ]
  }
}
```

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": "MoveI"
      },
      {
        "playerID": 2,
        "card": "Again"
      }
    ]
  }
}
```

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": 42,
    "x": 4,
    "y": 2
  }
}
```

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": 42,
    "cards": [
      "Spam",
      "..."
    ]
  }
}
```

Sollten alle Spamkarten vergeben sein, fragt der Server **vorher** nach der Auswahl des Spielers und schickt **nach der Wahl des Spielers** die Schadensmeldung.

```
{
  "messageType": "PickDamage",
  "messageBody": {
    "count": 2
  }
}
```

```
{
  "messageType": "SelectedDamage",
  "messageBody": {
    "cards": [
      "Virus",
      "Trojan"
    ]
  }
}
```

Schießende Roboter
Für Animationen hilfreich, ansonsten redundant.

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

Neustart
Ein Neustart benötigt keine genaueren Details. Zur Vereinfachung werden Roboter immer nach Oben / Norden ausgerichtet.

```
{
  "messageType": "Reboot",
  "messageBody": {
    "playerID": 42
  }
}
```

Drehung
Die Richtung wird hier mit "right" (Uhrzeigersinn), bzw. "left" (gegen den Uhrzeigersinn) angegeben.

```
{
  "messageType": "PlayerTurning",
  "messageBody": {
    "playerID": 42,
    "direction": "left"
  }
}
```

Energie
Anpassungen an Energiewerten werden mit der neuen Summe übertragen.
Ein zusätzlicher Wert liefert die Info, woher die Veränderung kam. Sie können Client-seitig entscheiden, wie Sie damit umgehen wollen.

```
{
  "messageType": "Energy",
  "messageBody": {
    "playerID": 42,
    "count": 1,
    "source": "field"
  }
}
```

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

```
{
  "messageType": "CheckPointReached",
  "messageBody": {
    "playerID": 42,
    "number": 3
  }
}
```

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

```
{
  "messageType": "GameFinished",
  "messageBody": {
    "playerID": 42
  }
}
```

Memory Swap und Cache Memory
Gibt die Karten an, die der Spieler auf sein Deck legen möchte.

```
{
  "messageType": "DiscardSome",
  "messageBody": {
    "cards": [
      "Card1",
      "..."
    ]
  }
}
```

Boink

```
{
  "messageType": "Boink",
  "messageBody": {
    "orientation": "up"
  }
}
```

Bewegende Checkpoints
Wenn sich die Ziele bewegen, wird es spannender!

```
{
  "messageType": "CheckpointMoved",
  "messageBody": {
    "checkpointID": 1,
    "x": 6,
    "y": 9
  }
}
```