Protokoll Version 1.1

October 26, 2015

Der Typ jedes Paket ist durch das erste Byte eindeutig identifiziert, das wir im folgenden als den *Messagetype* bezeichnen. Wir verwenden im Folgenden die Abkürzungen KS (Knockserver), GS (Gameserver) und CL (Client).

Der Knockserver

Der Knockserver stellt die Verbindung des Clients mit dem Gameserver her. Dazu lauscht der Knockserver auf einem spezifizierten Port X (etwa als Argument übergeben). Das Protkoll des Knockservers ist denkbar einfach:

1. CL etabliert Verbindung auf Port X
2. KS schickt HELLO-Message:

byte	int
42	game_server_port Y

3. CL antwortet mit BYE-Message, in welche er noch seinen Name an den Knockserver übermittelt (man ist ja höflich)

byte	int	$x\times$ byte
3	name_length x	name

Wird das Protokoll erfolgreich abgeschlossen, so reicht der Knockserver den Client weiter an den Gameserver. Um Nachrichten vom Gameserver zu erhalten, muss der Client dazu über Port Y (Host identisch zum Knockserver, in unserem Fall also *localhost*) eine Verbindung zum Gameserver herstellen.

Der Gameserver

Der Gameserver wartet so lange, bis genügend Clients sich per Knockprotokoll verbunden haben. Dann initiiert er das Spiel mit folgender Nachricht:

byte	int	int	$(w \times h) \times$ byte	byte
1	board_width w	board_height h	tile_types	free_tile

int	$2k \times$ int	int	$2l \times$ int	int	$2m \times$ int
#unmovable_tiles k	unmovable_x_y	#objectives l	objective_x_y	#players m	player_x_y

int	int	$n \times$ int
player_id	#player_objectives n	objective_id

Dabei ist der schwarz gefärbte Teil der Nachricht privat, d.h. für jeden Spieler individuell. Nach dem Versand dieser Nachricht wartet der Server einige Zeit (momentan etwa eine Sekunde) und beginnt dann das Spiel. Dazu fordert er einen Spieler mit folgender Nachricht zum Zug auf:

byte	long
3	timeout_in_ms

Der Spieler muss innerhalb des vorgegebenen Zeitlimits sich für einen Zug entscheiden, welchen er dann mittels

byte	int	byte	int	byte	2 × int	int
5	player_id	insert	insert_coord	tiletype	move_x_y	0

dem Server mitteilt. Folgende Felder benötigen eine weitergehende Erklärung:

- `insert`: Ein Bitfeld, in dem das erste Bit angiebt, ob die freie Kachel horizontal (1) oder vertikal (0) eingefügt wird und das zweite Bit, ob dabei von links oder rechts (1, 0) bzw. von oben oder unten (1,0) eingeschoben wird.
- `insert_coord`: Die x- bzw. y-Koordinate, an der die freie Kachel eingeschoben wird
- Die angehängte 0: Diese Nachricht wird, wie wir weiter unten sehen werden, für einen weiteren Zweck benutzt, dabei werden noch einige Informationen angehängt—sie ist damit nur notwendig, um den Programmiern des Servers weitere Arbeit zu ersparen.

Sollte dieser Zug gültig und fristgerecht beim Server ankommen, so broadcastet dieser eine leicht veränderte Variantes des obigen Pakets:

byte	...	2 × int	int
5	(siehe oben)	move_x_y	#coll_events n

n {

int	int	r × int
player_id	#collected r	objective_id

int	int	r × int
player_id	#collected r	objective_id

⋮

int	int	r × int
player_id	#collected r	objective_id

Das Feld `#coll_events` ist dabei genau die kuriose 0 des zuletzt beschriebenen Pakets. Der Zweck der angehängten Daten besteht darin, den Spielern mitzuteilen, welche Spieler gerade Zielvorgaben erfüllt haben—diese Information ist ja in Gänze nur dem Server bekannt.

Sollte ein Spieler entweder sein Zeitlimit überschreiben oder aber einen ungültigen Zug versuchen, so schmeißt der Server in aus dem Spiel hinaus. Dies

geschieht mit der folgenden garstigen Nachricht, die der Server an allen Spielern broadcastet:

byte	int	byte
4	player_id	reason

Netterweise liefert der Server dabei auch noch den Grund für den Spielverweis. Folgende Werte für das Feld `reason` sind im Protokoll vorgesehen:

Wert	Grund
0	Timeout wurde überschritten
1	Nachricht hat ein ungültiges Format
2	Netzwerkfehler (für die localhost-Spiele noch uninteressant)
3	Ungültiger Zug

Endet das Spiel, so sendet der Server eine Nachricht, in der die Gewinner benannt werden. Das Format ist wie folgt:

byte	int	$n \times$ int
2	#winners n	winner_id

Wir in den Regeln beschrieben, können mehrere Spieler gleichzeitig ihre (verbleibenden) Zielvorgaben einsammeln, daher kann es mehr als einen Gewinner geben. Sollte das Spiel enden, weil sich alle Spieler disqualifiziert haben, so ist die Liste der Gewinner schlicht leer.

Zuschauermodus

Neben den Spieler-Clients akzeptiert der Server auch Zuschauer. Der Unterschied zu dem obigen Protokoll ist zunächst, daß dem Knockserver kein Spielernamen übermittelt wird (`name_length` ist also 0). Außerdem sollte ein Zuschauer die Namen der Spieler wissen, daher schickt der Server nach der obigen Startnachricht (diejenige mit der Kennung 1) noch folgende Nachricht:

byte	int
6	#players n

n ⎧

int	$l_1 \times$ byte
name_length l_1	name

⋮

int	$l_n \times$ byte
name_length l_n	name

Dabei ist es egal, wann der Zuschauer sich anmeldet: der Server speichert alle bis dahin versendete Nachrichten in einem Backlog und übermittelt sie in der richtigen Reihenfolge. Anschließend werden die Nachrichten an die Zuschauer

genau wie an die Spielern verschickt (natürlich erhält ein Zuschauer nie die “Du-bist-am-Zug”-Nachricht).