

# Übungsblatt 12

Abgabe: 25.01.2026

---

Auf diesem Übungsblatt sollen die Anfänge für eine Mehrspieler-Variante unseres Spiels gelegt werden. Tatsächlich wird einfach die Bewegung der Spielfigur in einer Instanz des Spiel in einer zweiten, die per Netzwerk angebunden ist, repliziert. Dabei wird angenommen, dass es im Spiel keinen Zufall gibt, so dass sich beide Instanzen identisch verhalten, wenn die Spielfigur identische Aktionen ausführt. Die Netzwerkverbindung wird dabei zwischen jeweils einer Instanz der Klassen *RemotePlayer* und *ControlledPlayer* aufgebaut.<sup>1</sup> Zum Testen des Codes müsst ihr eine Kopie eures ganzen Projekts machen, damit ihr das Spiel zweimal aus zwei verschiedenen Instanzen von BlueJ starten könnt.

## Aufgabe 1 Fernsteuernd (40 %)

Leitet eine Klasse *RemotePlayer* von der Klasse *Player* ab. Diese soll in ihrem Konstruktor einen Socket zu einer Instanz der Klasse *ControlledPlayer* öffnen (s. [Aufgabe 2](#)).<sup>2</sup> Überschreibt die Methode *act* so, dass sie erst das *act* aus *Player* ausführt und danach die gewählte Bewegung an den *ControlledPlayer* überträgt. Ruft nach dem Senden die Methode *flush()* des Ausgabestroms auf, damit die Daten wirklich sofort gesendet werden. Überschreibt zusätzlich die Methode *setVisible* aus *GameObject* und schließt den Socket, wenn die Figur unsichtbar wird.<sup>3</sup>

## Aufgabe 2 Ferngesteuert (40 %)

Leitet eine Klasse *ControlledPlayer* von der Klasse *Player* ab. Diese soll in ihrem Konstruktor einen Server-Socket öffnen und eine einzige Verbindung akzeptieren, über die eine Instanz dieser Klasse fernsteuerbar ist.<sup>4</sup> Überschreibt dann die Methode *act*, so dass sie eine Bewegungsrichtung aus dem Socket liest und diese Richtung ausführt. Geht für das Schließen des Sockets ähnlich vor wie in [Aufgabe 1](#).

## Aufgabe 3 Spielend (20 %)

Ändert nun eure Klasse *Level* so ab, dass sie wahlweise einen *ControlledPlayer* oder einen *RemotePlayer* als Spielfigur erzeugen kann. Erweitert zudem die Hauptmethode eures Spiel, so dass ihr ihr übergeben könnt, welche Variante erzeugt werden soll. Wenn ihr flexibel sein wollt, können hier z.B. die IP-Adresse (bzw. keine Adresse für den *ControlledPlayer*) und der Port übergeben werden.

**Abgabe.** Analog zu Übungsblatt 6 und 10, d.h. im selben Repository in einem Ordner *loesung12*.

---

<sup>1</sup>Für eine echte Multispieler-Funktionalität müsste die Verbindung eher auf einer höheren Ebene angesiedelt und die ausgetauschten Daten auf die Spielobjekte verteilt werden.

<sup>2</sup>Wenn euer Spiel mehrere Level hat, solltet ihr mehrere Versuche mit kurzer Wartezeit dazwischen zulassen, weil es sein kann, dass der Server-Port noch nicht offen ist, wenn sich der Client anmelden möchte.

<sup>3</sup>Wenn die Figur in eurem Spiel schon vor dem Ende eines Levels unsichtbar werden kann, implementiert ihr stattdessen eine Methode *close*, die separat aufgerufen werden muss.

<sup>4</sup>Der Server-Socket kann nach dem Akzeptieren bereits wieder geschlossen werden.

## Aufgabe 4 Spiel veröffentlichen (Bonus 20 %)

In dieser Aufgabe geht es darum, ein schönes Spiel implementiert zu haben. Das Spiel muss auf der *Game/GameObject*-Basis beruhen, braucht aber nicht die Lösungen aller Übungsblätter zu enthalten, insbesondere nicht unbedingt die Netzwerkfunktionalität aus diesem Blatt.

Für die Veröffentlichung der Spiele wird das Informatik-GitLab verwendet. Die Spiele sind somit für alle sichtbar, die dazu einen Zugang haben. Ihr könnt euch auch entscheiden, euer Spiel für andere unsichtbar „zu veröffentlichen“. Dann sehen nur die Tutor:innen, dass ihr das gemacht habt.

Stattet euer Spiel mit einer richtigen *main*-Methode aus und exportiert es mit *Datei/Als jar-Archiv speichern...* Wählt dort eure Klasse mit der *main*-Methode aus und entfernt sonst alle Kreuze, bevor ihr auf *Weiter* klickt. Die exportierte Datei sollte sich per Doppelklick (Windows, macOS) oder im Terminal mit `java -jar <jar-Datei>` ausführen lassen, wenn eine Java-Runtime-Umgebung (JRE) installiert ist. Notfalls könnt ihr auch das Java verwenden, das sich im BlueJ-Ordner verbirgt.

Öffnet <https://gitlab.informatik.uni-bremen.de/roeper/pi1-spiele2025> und klickt darin oben auf + und dann auf *New Issue*. Gebt unter *Title* den Titel eures Spiels an und füllt die *Description* wie dort beschrieben aus (s. Abb. 1). Ihr könnt beliebig oft zwischen Editiermodus und dem *Preview* hin und her wechseln. Alternativ könnt ihr auch links unten in den Rich-Text-Modus wechseln. Wenn ihr euer Spiel vor anderen verbergen möchtet, kreuzt ihr *Turn on confidentiality...* an. Die weiteren Auswahlfelder könnt ihr ignorieren. Klickt dann auf *Create Issue*.

Damit andere eine Chance haben, sich euer Spiel vor der Wahl des besten Spiels anzusehen, sollte es bis zum späten Nachmittag des Mittwochs, den 28.01.2026 hochgeladen sein. Ihr könnt es später auch noch ersetzen. Die formale Abgabefrist, um die Punkte für diese Aufgabe zu erhalten, ist der 31.01.2026.

## Aufgabe 5 Spiele bewerten

Ihr könnt euch jederzeit alle Spiele ansehen<sup>5</sup> und auch *Likes* vergeben. Die Liste der Issues kann nach verschiedenen Kriterien sortiert werden, wenn ihr im Menü links *Issues* und dann *List* auswählt. *Dislikes* sind nicht erlaubt, denn es ist ja bereits eine tolle Leistung, es bis zum fertigen Spiel gebracht zu haben. Aus den Spielen mit den meisten Likes wird in der letzten Vorlesung das Gewinnerspiel ermittelt.


---

<sup>5</sup>Da die jar-Dateien aus dem Internet stammen, müsst ihr euer Betriebssystem davon überzeugen, sie auszuführen.


Write
Preview

## PI-1-Spiel

Thomas Röfer



Die Figur wird mit den Pfeiltasten gesteuert und muss das Ziel erreichen, ohne dabei den anderen Personen zu nahe zu kommen. Leider endet das Spiel nur, wenn sie sich tatsächlich einer Person zu stark nähert. Wird das Ziel erreicht, passiert leider nichts.

 [Das Spiel zum Herunterladen](#)

## Lizenzen

### Grafiken

Die Grafiken stammen von verschiedenen Webseiten. Unter den angegebenen URLs finden sich die Originale, die Namen der Urheber und die Verweise auf die verwendeten Lizenzen:

- Hintergrundgrafiken: <https://opengameart.org/content/rpg-tiles-cobble-stone-paths-town-objects>, CC-BY-SA 3.0. Aus der Grafik PathAndObjects.png wurde einzelne 32x32-Pixel- Kacheln ausgeschnitten und jeweils zu 3x3-Kacheln kombiniert.
- Figuren: [http://allacrost.org/wiki/index.php/Artwork\\_Categories](http://allacrost.org/wiki/index.php/Artwork_Categories), CC-BY-SA 4.0. Aus den Grafiken Sprite\_claudius\_walk.png, Sprite\_old\_woman\_walk.png, Sprite\_male\_child\_walk.png und Sprite\_laila\_run.png wurden die Ansichten der ersten Spalte als 32x64-Pixel-Grafiken ausgeschnitten, unten um 16 transparente Pixelreihen erweitert und jeweils als separate Grafiken gespeichert.
- Ziel: <https://davidjakubec.itch.io/64-x-64-sprite-sheet>, "completely free". Aus der Grafik spriteSheet.png wurde das Ziel-Schild ausgeschnitten und als separate Grafik gespeichert.

### Sounds

Die Sound stammen von freesound.org. Unter den angegebenen URLs finden sich die Originale, die Namen der Urheber und die Verweise auf die verwendeten Lizenzen:

- step.wav: <http://freesound.org/people/colo777/sounds/251435>, CC0. Gekürzt.
- error.wav: <https://freesound.org/people/Autistic%20Lucario/sounds/142608/>, CC-BY 3.0
- go-away.wav: <https://freesound.org/people/pyro13djt/sounds/331908/>, CC0, Gekürzt.

Abbildung 1: Wie ein *Preview* zur Beschreibung eines Spiels aussehen könnte.