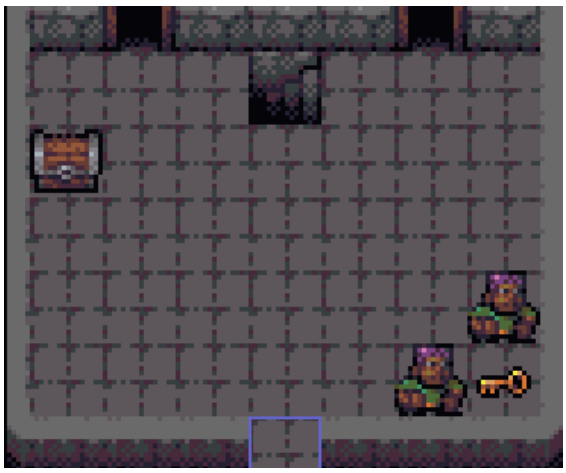
Synchronisationsprobleme/Race Conditions im Dungeon Code

Unser Code funktioniert, wenn wir das Spiel als Einzelspieler spielen. Er ist aber fehlerhaft, und diese Fehler sind nur schwer zu sehen, weil sie nur als Race Conditions in Fällen auftreten, wo mehrere Spieler fast gleichzeitig eine Aktion ausführen. Nimm beispielsweise folgende Spielsituation, in welcher "Berserker" und "Dawnrider" beide am Schlüssel in der Ecke des Raumes interessiert sind:



Was passiert, wenn die beiden Spieler beide praktisch gleichzeitig den Schlüssel aufnehmen wollen? Das Aufnehmen eines Objekts wird in dungeon_game.py über folgenden Code gesteuert:

```
def on_object_taken(gameworld, object_position, object_id):
    storage.add_object_to_player_inventory(gameworld.player_id, object_id)
    storage.remove_object_from_room(gameworld.room_id, object_position)
```

Es gibt also 2 Schritte, die im Storage Backend ausgeführt werden, um ein Objekt zu nehmen: Zuerst wird das Objekt in das Inventar des Spielers aufgenommen, dann wird es aus dem Raum entfernt. Den entsprechenden Code siehst du hier:

```
def add_object_to_player_inventory(playerid, objectid):
    """
    Adds the given object ID to the inventory of the given player.
    """
    cur = connection.cursor()
    QUERY = "INSERT INTO inventory (playerid, objectid) VALUES (?, ?)"
    cur.execute(QUERY, (playerid, objectid,))
    connection.commit()
```

```
def remove_object_from_room(roomid, tileid):
    """
    Removes any object from the given tile in the given room.
    """
    cur = connection.cursor()
    parameters = {
        'objectindex': tileid,
        'roomid': roomid
        }
    cur.execute("DELETE FROM ObjectMap WHERE objectindex = :objectindex AND roomid = roomid ", parameters)
    connection.commit()
```

Beide Funktionen schliessen jeweils eine Transaktion mit einem Commit ab, d.h. beide Aktionen laufen im Backend jeweils als separate Transaktion. Das Backend kann uns deshalb nicht garantieren, dass diese Aktionen entweder beide gelingen oder beide nicht ausgeführt werden.

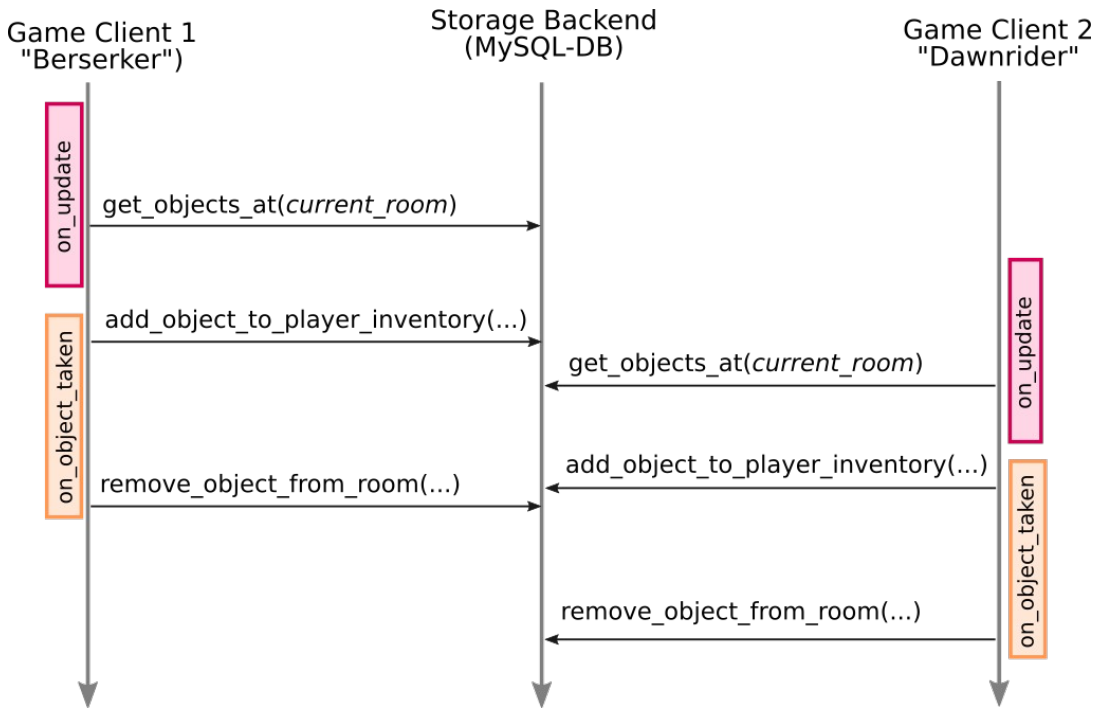
Es gibt noch ein weiteres Problem. Das Storage Backend ist die "Single Source of Truth" in unserem Spiel. Das heisst, es ist der Zustand der Datenbank, nicht der Zustand einer einzelnen Spiel-Instanz, welcher definiert, wo sich welche Objekte im Spiel befinden bzw. wer sie aufgenommen hat.

Jede Spiel-Instanz muss sicherstellen, dass ihr Zustand mit dem tatsächlichen Zustand übereinstimmt. In dungeon_game.py wird dies gelöst, indem in `on_update` ungefähr 10 Mal pro Sekunde mit dem Zustand des Backends synchronisiert wird:

```
time_count = 0
def on_update(dt):
    global time_count
    # dt is in ms (milliseconds)
    time_count += dt
    if time_count < 100:
        return

    time_count = 0
    # do this every 100 ms (eg 10 times per second)
    players = storage.get_players_at(gameworld.room_id)
    objects = storage.get_objects_at(gameworld.room_id)
    gameworld.set_players(players)
    gameworld.set_objects(objects)
```

Nun kann es zu folgender ungünstiger Verzahnung von Interaktionen mit dem Storage Backend kommen:



Der bestehende Code hat deshalb mehrere Synchronisationsprobleme:

- Die Dawnrider-Spielinstanz *kann nicht wissen*, dass das Objekt, welches der Spieler aufnehmen will, nicht mehr da ist, weil die Berserker-Spielinstanz das Objekt erst aus dem Raum entfernt, *nachdem* Dawrnrider's `on_update` die Liste der Objekte im Raum auf den neusten Stand gebracht hat. Deshalb kann es nicht verhindern, dass sein `on_object_taken` ausgeführt wird.
- `on_object_taken` wird nicht atomar ausgeführt (d.h. entweder gelingen alle Aktionen oder keine). Deshalb wird der Schlüssel sowohl in Berserkers wie auch in Dawnriders Inventar aufgenommen; Dawnriders `remove_object_from_room` bewirkt nichts, weil ein Datensatz aus einer Tabelle entfernt werden soll, der gar nicht mehr exisitert, aber das hat keine Auswirkung mehr darauf, dass der Schlüssel bereits in Dawnriders Inventar aufgenommen wurde.