



Simutrans Squirrel 3.2

Script-API-Migration, Kompatibilität und Entwicklerreferenz

Produkt	Simutrans Standard	Squirrel-Runtime	3.2 (SQUIRREL_VERSION_NUMBER 320)
Integrationsrevisionen	r12165 – r12168 (trunk)	Abschlussrevision	r12168
Dokumentationsstand	Aktuell zum Stand der Integrationsrevision	Dokumentversion	1.0
Datum	2026-08-16	Dokumentart	Entwicklerreferenz
Sprache	KI-unterstützte Übersetzung — menschliche Prüfung ausstehend		
Zielgruppe	Skriptautoren · Szenarioautoren · KI-Autoren · Simutrans-Entwickler		

Erstellt auf Grundlage der validierten Squirrel-3.2-Migration in Simutrans Standard.

Hinweis. KI-unterstützte Übersetzung — menschliche Prüfung ausstehend. Die englische Fassung ist die kanonische technische Referenz.

ZUSAMMENFASSUNG

Simutrans Standard bettet jetzt **Squirrel 3.2** ein. Die eingebetteten Squirrel-Quellen wurden von einem Stand der Entwicklungslinie 3.1.x aus dem Jahr 2021 auf die Upstream-Veröffentlichung **v3.2** aktualisiert.

Die wichtigen Punkte für alle, die Simutrans-Skripte schreiben oder pflegen:

- Dies ist eine Aktualisierung der **Sprach-Runtime**, keine Neugestaltung der Simutrans Script API.
- Die Migration wurde bewusst so angelegt, dass sie die **öffentliche Script API erhält**. Durch die Runtime-Aktualisierung wurde keine registrierte Funktion hinzugefügt, entfernt oder umbenannt, und es hat sich keine Parameterliste und kein Verhalten von Standardargumenten geändert.
- **Vorhandene kompatible Szenarien und KI-Skripte müssen nicht grundlegend neu geschrieben werden.**
- Mehrere **interne VM-Integrationspunkte** haben sich geändert, vor allem der C++-Aufruf zum Nachschlagen von Instanzen, die Closure-Erzeugung für die Umgebungsbindung und die String-Hash-Funktion.
- Für Skripte gibt es **einen beobachtbaren semantischen Unterschied**: die Reihenfolge, in der die Einträge einer Tabelle besucht werden. Er ist in Abschnitt 6 dokumentiert und ist das Einzige, dessen Überprüfung sich in einem vorhandenen Skript lohnt.
- **Alte Spielstände mit Skriptzustand wurden ausdrücklich getestet** und werden korrekt geladen.
- Mit der Runtime kamen zwei neue Erleichterungen in Sprache und Bibliothek hinzu: eine Inline-Form der Umgebungsbindung und `table.map`.

Wenn Sie ein Skript pflegen und nur einen Abschnitt dieses Dokuments lesen, lesen Sie [Abschnitt 6](#).

Inhalt

1.	Zusammenfassung	1
2.	Geltungsbereich und Begriffe	3
3.	Was sich geändert hat	3
4.	Was sich nicht geändert hat	5
5.	Kompatibilitätsmatrix	5
6.	Iterationsreihenfolge von Tabellen	6
7.	Migrationsleitfaden für Skriptautoren	7
8.	Szenarioautoren	8
9.	KI-Autoren	9
10.	Paksets	9
11.	Beispiele	10
12.	Hinweise für Simutrans-Entwickler: die Einbettungsschicht	11
13.	Stabilität der Script API	12
14.	Zusammenfassung der Validierung	13
15.	Bekannte Einschränkungen und Anmerkungen	14
16.	Wo was zu finden ist	14
17.	Verfügbarkeit und Änderungsprotokoll	15
18.	Dokumenthistorie	16

2. Geltungsbereich und Begriffe

Dieses Dokument verwendet drei Begriffe in einem genauen Sinn, und die Unterscheidung ist durchgehend von Bedeutung:

Begriff	Bedeutung
Squirrel	Die eingebettete <i>Skriptsprache und ihre Runtime</i> , ein externes Projekt, das in Simutrans mitgeführt wird.
Script API	Die eigene registrierte Schnittstelle von Simutrans — die Klassen, Methoden und Konstanten, die Simutrans für Skripte bereitstellt, dokumentiert in der Script-API-Referenz.
Simutrans Standard	Die Hauptlinie von Simutrans, im Unterschied zu Simutrans Extended.

„Squirrel 3.2“ bezeichnet also die Runtime. Es bezeichnet keine neue Version der Script API; diese wird getrennt versioniert und ist von dieser Arbeit unverändert.

Herkunft der Runtime

Die mit Simutrans ausgelieferten Squirrel-Quellen sind eine mitgeführte Kopie des Upstream-Projekts unter [albertodemichelis/squirrel](https://github.com/albertodemichelis/squirrel).

	Commit	Beschreibung
Bisherige mitgeführte Grundlage	<code>23a0620658714b996d20da3d4dd1a0dcf9b0bd98</code>	Ein Stand der Entwicklungslinie 3.1.x vom 2021-09-16
Neue Grundlage	<code>f92bc298784ceea459b12e2de33bdf672bfeb83</code>	Upstream-Release-Tag <code>v3.2</code> vom 2022-02-10

Der bisherige Stand stammte aus der Entwicklungslinie *nach* der Veröffentlichung von 3.1 und war daher weder ein exaktes Upstream-3.1 noch ein 3.2. Die Aktualisierung schlicht als „3.1 auf 3.2“ zu beschreiben wäre ungenau; deshalb werden stattdessen die obigen Commits angegeben.

3. Was sich geändert hat

3.1 Überblick

Änderung	Für Skripte sichtbar?	Für C++-Entwickler sichtbar?
Mitgeführte Squirrel-Quellen auf Upstream v3.2 aktualisiert	mittelbar	ja

Änderung	Für Skripte sichtbar?	Für C++-Entwickler sichtbar?
Instanz-Lookup-API hat einen Parameter erhalten	nein	ja
Inline-Syntax für <code>bindenv</code> zur Sprache hinzugefügt	ja, als neue Möglichkeit	ja
<code>table.map</code> zur Standardbibliothek hinzugefügt	ja, als neue Möglichkeit	nein
String-Hash-Funktion geändert	ja — siehe Abschnitt 6	ja

3.2 Instanz-Lookup-API

Upstream-Squirrel 3.2 hat die Signatur derjenigen C-API-Funktion geändert, die einen nativen Instanzzeiger vom VM-Stack zurückholt. Sie hat einen fünften Parameter erhalten, der steuert, ob eine Typabweichung einen Squirrel-Fehler auslöst oder dem Aufrufer als einfacher Fehlschlag gemeldet wird:

```
/* before */
SQLRESULT sq_getinstanceup(HSQIRRELVM v, SQInteger idx, SQUserPointer *p, SQUserPointer typetag);

/* Squirrel 3.2 */
SQLRESULT sq_getinstanceup(HSQIRRELVM v, SQInteger idx, SQUserPointer *p, SQUserPointer typetag,
SQBool throwerror);
```

Jede Aufrufstelle in Simutrans wurde so angepasst, dass sie den Wert übergibt, der ihr bisheriges Verhalten erhält. **Skriptautoren sind nicht betroffen:** Dies ist eine C++-Signatur und kein Einstiegspunkt der Script API, und das beobachtbare Verhalten der betroffenen Aufrufe ist unverändert.

Berührte Aufrufstellen in Simutrans: `src/simutrans/script/api/api_map_objects.cc`, `src/simutrans/script/api_class.h`, `src/squirrel/sq_extensions.cc`, dazu die mitgeführten Dateien der Standardbibliothek.

3.3 `bindenv` und die Inline-Form der Umgebung

Eine Umgebung an eine Funktion zu binden ist ein Merkmal der Sprache Squirrel, keine Simutrans-Klasse und keine Registrierung in der Script API.

Die vorhandene Methode `bindenv` ist **unverändert und funktioniert weiterhin**. Squirrel 3.2 fügt eine Inline-Form hinzu, bei der die Umgebung in eckigen Klammern zwischen dem Schlüsselwort `function` und der Parameterliste geschrieben wird. Beide Formen sind in [Abschnitt 11](#) gezeigt.

Die Inline-Form ist eine Erleichterung. Sie gibt keinen Zugriff auf etwas, das die Methodenform nicht ohnehin schon erlaubte, und **vorhandene Skripte brauchen keine Änderung**.

Intern erforderte dies, dass der Opcode zur Closure-Erzeugung das Bindungsziel mitführt; daher wurden sowohl der Compiler als auch die VM angepasst.

3.4 `table.map`

Squirrel 3.2 fügt Tabellen in der Standardbibliothek eine Methode `map` hinzu. Arrays hatten bereits eine.

Zwei Eigenschaften sollte man vor der Verwendung kennen:

- `table.map` liefert ein **Array** der Callback-Ergebnisse zurück, keine Tabelle. Darin unterscheidet es sich von `table.filter`, das eine Tabelle zurückgibt.
- Die Ergebnisse erscheinen in der Iterationsreihenfolge der Tabelle; die Reihenfolge des zurückgegebenen Arrays unterliegt also [Abschnitt 6](#). Sortieren Sie es, wenn die Reihenfolge von Bedeutung ist.

3.5 String-Hash-Funktion

Upstream hat die auf String-Schlüssel angewandte Hash-Funktion geändert. Dies ist die eigentliche Ursache des einen für Skripte sichtbaren semantischen Unterschieds dieser Migration; er wird ausführlich in [Abschnitt 6](#) behandelt.

4. Was sich nicht geändert hat

Dieser Abschnitt ist bewusst ausdrücklich gehalten, denn er ist der Teil, den die meisten Skriptautoren brauchen.

- **Die Simutrans Script API ist unverändert.** Kein Klassenname hat sich geändert. Keine öffentliche Methode hat sich geändert. Kein Registrierungsname hat sich geändert. Keine Parameterliste und kein Verhalten von Standardargumenten hat sich geändert. Das wurde gemessen, nicht angenommen — siehe [Abschnitt 13](#).
- **Die Einstiegspunkte für Szenarien und KI sind unverändert.**
- **Das Spielstandformat ist unverändert.**
- **Die Art, wie Skripte geladen, angehalten und fortgesetzt werden, ist unverändert.**
- **Keinerlei Paket-Änderung ist erforderlich** — siehe [Abschnitt 10](#).

5. Kompatibilitätsmatrix

Bereich	Kompatibilitätsstand	Maßnahme für Entwickler
Vorhandene Namen der Script API	Unverändert. Anzahl der Registrierungen und Aufrufkonventionen vorher und nachher als identisch gemessen.	Keine.
Szenarioskripte	Kompatibel , vorbehaltlich des Hinweises zur Iterationsreihenfolge.	Auf einem aktuellen Build testen; jede Tabelleniteration prüfen, die eine Auswahl trifft.
KI-Skripte	Kompatibel. Die beiden mitgelieferten skriptgesteuerten KI-Spieler wurden gegen die neue Runtime ausgeführt und blieben kompatibel.	Wie bei Szenarien. Damit rechnen, dass Zeitplanung und Streckenwahl im Detail abweichen — siehe Abschnitt 9.
Spielstände mit Skriptzustand	Alte Spielstände werden korrekt geladen , getestet mit den mitgelieferten KI-Spielern. Speichern und erneutes Laden unter 3.2 wurde ebenfalls getestet.	Keine. Vor umfangreichen Tests wie immer sichern.

Bereich	Kompatibilitätsstand	Maßnahme für Entwickler
Umgekehrte Richtung: 3.2-Spielstand in einem älteren Build	Nicht zugesichert. Nicht Teil dieser Arbeit und nicht getestet.	Verlassen Sie sich nicht darauf.
VM-Anbindungen (C++-Einbettungsschicht)	Geändert. Das Instanz-Lookup hat einen Parameter erhalten; die Closure-Erzeugung führt ein Bindungsziel mit.	Betrifft nur Code, der <code>src/squirrel/</code> oder die Registrierungsschicht der Script API berührt. Siehe Abschnitt 12.
Iterationsreihenfolge von Tabellen	Geändert. Andere String-Hash-Funktion, daher eine andere <code>foreach</code> -Reihenfolge.	Jedes Skript prüfen, das sich auf die Reihenfolge verlässt, die eine Tabelle zufällig liefert. Siehe Abschnitt 6.
Iterationsreihenfolge von Arrays	Unverändert. Arrays sind geordnete Container; ihre Reihenfolge gehört zu ihrem Vertrag.	Keine.
Fehlerdiagnostik	Keine neuen Squirrel-Fehler beobachtet in der Validierungskampagne.	Keine.
Pakset-Inhalte (<code>.dat</code> , Objekte, Grafiken)	Nicht betroffen.	Keine.

Hinweis zum Geltungsbereich. Die obigen Kompatibilitätsaussagen beschreiben die Skripte und Codepfade, die von der Validierungskampagne abgedeckt wurden — die mit Simutrans ausgelieferten skriptgesteuerten KI-Spieler und die aufgeführten Richtungen beim Laden und Speichern von Spielständen. Sie sind keine Zusicherung, dass jedes denkbare Skript von Dritten unbetroffen ist.

6. Iterationsreihenfolge von Tabellen

ACHTUNG

Verlassen Sie sich nicht auf die Iterationsreihenfolge von Squirrel-Tabellen, sofern sie nicht ausdrücklich von der API dokumentiert ist.

Squirrel 3.2 ändert die für String-Schlüssel verwendete Hash-Funktion. Tabellen sind Hash-Container, daher ändert sich dadurch die Reihenfolge, in der `foreach` ihre Einträge besucht.

6.1 Was das tatsächlich bedeutet

Die Folge für ein Skript ist mittelbar, aber real. Wenn ein Skript eine Tabelle durchläuft, um das nächste zu bearbeitende Element auszuwählen, kann die neue Reihenfolge ändern:

- welcher Eintrag zuerst betrachtet wird,
- in welcher Reihenfolge Arbeit eingeplant wird,
- welcher Kandidat gewinnt, wenn mehrere gleich gut sind,

- und damit, zu welchem von mehreren gleichermaßen gültigen Ergebnissen das Skript gelangt.

6.2 Was es nicht bedeutet

Das ist keine Beschädigung und keine Zufälligkeit. Bei gleicher Eingabe verhält sich die Runtime jedes Mal gleich. Das Skript behält seinen vollständigen logischen Zustand, und die Ausführung bleibt gültig.

Geändert hat sich eine Entscheidung bei Gleichstand, die von vornherein nie zugesichert war.

6.3 Was dagegen zu tun ist

Wenn ein Skript eine festgelegte Reihenfolge braucht, muss es sie selbst herstellen, statt sich auf die Reihenfolge zu verlassen, die eine Tabelle zufällig liefert. Sammeln Sie die Schlüssel und sortieren Sie sie:

```
// Fragile: the order depends on the runtime's hash function.
foreach (key, value in prices) {
    consider(key, value)
}

// Defined: the order is the script's own choice.
local keys = prices.keys()
keys.sort()
foreach (key in keys) {
    consider(key, prices[key])
}
```

SQUIRREL

Sowohl `table.keys()` als auch `array.sort()` gehören zu den Squirrel-Standarddelegates, die in Simutrans verfügbar sind.

Arrays sind nicht betroffen: Sie sind geordnete Container, und ihre Reihenfolge gehört zu ihrem Vertrag. **Bevorzugen Sie Arrays, wo die Reihenfolge Bedeutung trägt.**

6.4 Gespeicherter Skriptzustand

Wenn der persistente Zustand eines Skripts erneut geschrieben wird, können die Einträge einer gespeicherten Tabelle in einer anderen Reihenfolge als zuvor erscheinen. Der gespeicherte Zustand selbst ist davon nicht betroffen: Er wird **nach Namen wiederhergestellt, nicht nach Position.**

7. Migrationsleitfaden für Skriptautoren

7.1 Wenn Ihr Skript bereits funktioniert

Es gibt keinen Migrationsschritt. Konkret:

1. **Testen Sie es auf einem aktuellen Build.** Führen Sie das Szenario oder die KI wie gewohnt aus.
2. **Prüfen Sie Warnungen und Fehler zur Laufzeit.** Die Validierungskampagne hat keine neuen Squirrel-Fehler erzeugt; wenn Ihr Skript einen erzeugt, lohnt sich eine Meldung.
3. **Prüfen Sie Code, der eine Tabelle durchläuft, um eine Entscheidung zu treffen.** Wenn der erste Treffer gewinnt oder der letzte, entscheiden Sie, ob das beabsichtigt war, und machen Sie es ausdrücklich.
4. **Schreiben Sie nichts um, was nicht defekt ist.** Es hat sich keine API geändert.

Eine Warnung zur Testmethode: **Vergleichen Sie zwei Durchläufe nicht anhand ihrer Protokolle.** Zwei Durchläufe desselben Skripts unter verschiedenen Simutrans-Versionen können unterschiedliche, gleichermaßen gültige Entscheidungen treffen. Ein Protokollvergleich zeigt Rauschen, das kein Defekt ist.

7.2 Wenn Ihr Skript fehlschlägt

Arbeiten Sie diese Reihenfolge der Reihe nach ab. Sie ist darauf angelegt, die Ursachen rasch zu trennen.

1. **Erfassen Sie den tatsächlichen Squirrel-Fehler.** Die Meldung und die genannte Zeile sind der Ausgangspunkt; stellen Sie keine Diagnose allein anhand von Symptomen.
2. **Gleichen Sie die Verwendung der Script API mit der Referenz ab.** Die API hat sich bei dieser Migration nicht geändert; ein API-Fehler bedeutet daher meist, dass das Skript bereits etwas verwendet hat, das sich zu einem anderen Zeitpunkt geändert hatte oder nie korrekt war. Prüfen Sie das [Änderungsprotokoll der Script API](#) für die Version, auf die Sie abgezielt haben.
3. **Prüfen Sie Annahmen zur Iteration.** Suchen Sie nach `foreach` über eine Tabelle, bei dem das Ergebnis davon abhängt, welcher Eintrag zuerst kommt. Das ist die eine Kategorie, die diese Migration plausibel beeinflusst haben kann.
4. **Prüfen Sie auf Verhalten, das Sie von der alten Runtime übernommen haben.** Wenn sich das Skript auf eine Eigenheit des bisherigen mitgeführten Standes verlassen hat statt auf dokumentiertes Squirrel-Verhalten, ist das die wahrscheinlichste verbleibende Ursache.
5. **Trennen Sie Probleme der Script API von Annahmen über Pakset und Szenario.** Ein fehlendes Objekt, ein umbenannter Weg oder ein Pakset-Unterschied erzeugt Fehlschläge, die wie Skriptfehler aussehen, aber keine sind. Klären Sie, welchen Fall Sie vor sich haben, bevor Sie Code ändern.

HINWEIS

Wenn der Fehlschlag alle fünf Schritte übersteht und reproduzierbar ist, lohnt sich eine Meldung im Forum mit dem Fehlertext, der Simutrans-Revision und einem minimalen Skript, das ihn reproduziert.

8. Szenarioautoren

Erwartete Kompatibilität. Szenarien brauchen keine Änderungen. Die Einstiegspunkte für Szenarien, die Script API und das Spielstandformat sind alle unverändert.

Empfohlene Validierung.

- Laden Sie das Szenario auf einem aktuellen Build und spielen Sie weit genug, um seine Regeln und seine Gewinn- und Verlustbedingungen auszuüben.
- Speichern und laden Sie mitten im Szenario erneut und bestätigen Sie, dass der persistente Zustand unversehrt ist.
- Prüfen Sie jede Regel, die eine Tabelle von Fabriken, Haltestellen, Städten oder Spielern durchläuft, um ein Ziel auszuwählen oder etwas zuzuteilen. Wenn zwei Kandidaten gleich gut sein könnten, kann nun der andere gewinnen.

Semantische Vorbehalte.

- Die geänderte Iterationsreihenfolge beeinflusst, *welches* von mehreren gleichermaßen gültigen Ergebnissen eintritt, nicht *ob* das Szenario funktioniert. Ein Szenario, dessen Text „die erste gefundene

Fabrik" sagt, sollte vielleicht etwas Genaueres sagen.

- Wenn das erwartete Ergebnis eines Szenarios gegen einen festen Wert geprüft wird, der aus einer Tabelleniteration abgeleitet wurde, machen Sie diese Ordnung ausdrücklich, wie in Abschnitt 6.3 gezeigt.

9. KI-Autoren

Erwartete Kompatibilität. Die beiden mit Simutrans ausgelieferten skriptgesteuerten KI-Spieler, `sqai` und `sqai_rail`, wurden gegen die neue Runtime ausgeführt und blieben kompatibel. In der getesteten Kampagne bewahrten sie ihren logischen Verbindungszustand, machten weiterhin gültige Fortschritte und erzeugten keine Skript- oder Laufzeitfehler. Das Laden eines Spielstands, der von der bisherigen Runtime geschrieben wurde, wurde ebenfalls getestet und funktioniert.

Empfohlene Validierung.

- Führen Sie eine Kampagne von realistischer Länge aus statt eines kurzen Rauchttests. Unterschiede im KI-Verhalten summieren sich über die Zeit, und ein Zwei-Minuten-Lauf wird sie nicht zeigen.
- Bestätigen Sie, dass der persistente Zustand der KI einen Speicher-/Ladezyklus übersteht.
- Achten Sie auf eine KI, die keine Fortschritte mehr macht, nicht auf eine KI, die andere Fortschritte macht. Das Zweite ist zu erwarten.

Semantische Vorbehalte.

- **Entscheidungen bei Zeitplanung und Streckensuche können abweichen.** Das ist die erwartete Folge der geänderten Iterationsreihenfolge und kein Defekt.
- Eine KI, die ihre Kandidatenziele durchläuft und das erste annehmbare nimmt, wird nun möglicherweise ein anderes annehmbares nehmen. Wenn eine bestimmte Priorität von Bedeutung ist, sortieren Sie die Kandidaten ausdrücklich.
- Deshalb prüft ein KI-Test, der eine genaue Abfolge gebauter Verbindungen zusichert, die Hash-Funktion der Runtime und nicht die KI. Prüfen Sie stattdessen Eigenschaften — das Netz ist verbunden, die Bilanz ist positiv, es wurden keine Fehler ausgelöst.

10. Paksets

Diese Migration erfordert keinerlei Pakset-Änderungen.

Im Einzelnen:

- **Keine Migration des pak-Objektformats.** Das binäre pak-Format bleibt unangetastet.
- **Keine `.dat`-Migration.** Es wurde kein Parameter hinzugefügt, entfernt oder neu gedeutet.
- **Keine Grafikmigration.**
- **Keine Änderung an `menuconf.tab` oder an der Konfiguration.**

Skripte und Pakset-Inhalte sind getrennte Belange. Ein Pakset, das Szenarien ausliefert, ist nur über diese Szenarien betroffen, genau wie in Abschnitt 8 beschrieben, und nur im Sinne der Iterationsreihenfolge — der Pakset-Inhalt selbst braucht nichts.

11. Beispiele

Alle nachstehenden Beispiele sind gültiges Squirrel und verwenden nur Funktionen, die in Simutrans zum Stand der Integrationsrevision verfügbar sind.

11.1 Umgebungsbindung — beide Formen

Die vorhandene Methodenform ist unverändert und funktioniert weiterhin:

```
local env = { factor = 2 }
local scale = function(x) { return x * factor }
local bound = scale.bindenv(env)  // 'this' inside scale is now env
bound(21)                        // 42
```

SQUIRREL

Squirrel 3.2 fügt eine Inline-Form hinzu. Die Umgebung wird in eckigen Klammern zwischen dem Schlüsselwort `function` und der Parameterliste geschrieben:

```
local env = { factor = 2 }
local scale = function [env] (x) { return x * factor }
scale(21)                        // 42
```

SQUIRREL

Dieselbe Klammerform wird für benannte Funktionen, für lokale Funktionen sowie für Klassen- und Tabellenmitglieder einschließlich `constructor` akzeptiert. Beide Formen erzeugen eine Closure, deren Umgebung die angegebene Tabelle, Klasse oder Instanz ist.

Hinweis zur Portabilität. Die Inline-Form ist ein *neues Sprachmerkmal*. Ein Skript, das sie verwendet, läuft nicht auf älteren Simutrans-Versionen. Die Methodenform läuft überall.

11.2 `table.map`

```
local prices = { coal = 10, oil = 25 }
local labels = prices.map(function(key, value) {
    return key + "=" + value
})
```

SQUIRREL

Der Callback wird einmal je Eintrag aufgerufen und erhält den Schlüssel und den Wert, wobei `this` an die Tabelle gebunden ist. `labels` ist ein **Array**, und die Reihenfolge seiner Elemente folgt der Iterationsreihenfolge der Tabelle — sortieren Sie es, wenn das von Bedeutung ist.

Hinweis zur Portabilität. `table.map` ist ein *neues Merkmal der Standardbibliothek*. Ein Skript, das es verwendet, läuft nicht auf älteren Simutrans-Versionen.

11.3 Iteration mit festgelegter Reihenfolge

```
// Collect the keys, impose an order, then work through them.
local keys = prices.keys()
keys.sort()
foreach (key in keys) {
    do_something(key, prices[key])
}
```

SQUIRREL

Dieses Muster lohnt sich überall dort, wo die innerhalb der Schleife getroffene Auswahl im Ergebnis sichtbar ist — das Ziel, zu dem eine KI baut, die Fabrik, die ein Szenario auswählt, der Eintrag, den ein Bericht zuerst aufführt.

12. Hinweise für Simutrans-Entwickler: die Einbettungsschicht

Dieser Abschnitt richtet sich an Personen, die `src/squirrel/` oder die Registrierungsschicht der Script API berühren. Er ist eine Zusammenfassung dessen, was sich bewegt hat, kein Durchgang durch den Quelltext.

12.1 Instanz-Lookup

`sq_getinstanceup()` hat einen fünften Parameter erhalten, `SQBool throwerror`, der auswählt, ob eine Abweichung des Typ-Tags einen Squirrel-Fehler auslöst oder dem Aufrufer einen einfachen Fehlschlag zurückgibt.

Jede Aufrufstelle wurde so angepasst, dass ihr bisheriges Verhalten erhalten bleibt:

- `src/simutrans/script/api_class.h` und `src/simutrans/script/api/api_map_objects.cc` übergeben `SQTrue` und behalten damit den bestehenden Vertrag von Simutrans bei, dass ein falscher Typ ein Skriptfehler ist.
- `src/squirrel/sq_extensions.cc` übergibt `SQTrue`.
- Die mitgeführten Dateien der Standardbibliothek (`sqstdblob.cc`, `sqstdio.cc`, `sqstdstream.cc`, `sqstdstring.cc`) folgen Upstream, das `SQFalse` dort verwendet, wo ein fehlgeschlagenes Lookup lokal behandelt wird.

Wenn Sie eine neue Script-API-Klasse hinzufügen, folgen Sie `api_class.h`: Übergeben Sie `SQTrue`, damit eine Typabweichung dem Skriptautor als Squirrel-Fehler erscheint und nicht als stilles null.

12.2 `bindenv` und Closure-Erzeugung

`SQVM::CLOSURE_OP()` hat einen Parameter `boundtarget` erhalten, und der Compiler gibt ihn für die Inline-Form `function [env] (...)` aus. Nichts in der Registrierungsschicht der Script API hängt davon ab; es ist nur von Bedeutung, wenn Sie den Compiler oder die VM verändern.

12.3 String-Hashing

`_hashstr()` in `src/squirrel/squirrel/sqstring.h` wurde durch die Upstream-Fassung aus 3.2 ersetzt. Sie durchläuft den String in der entgegengesetzten Richtung und verwendet eine andere Schrittberechnung. Dies ist die Quelle der in Abschnitt 6 beschriebenen Änderung der Iterationsreihenfolge von Tabellen.

Praktische Folge für die Arbeit an der Engine: Alles, was eine Squirrel-Tabelle durchlaufen hat und von der sich ergebenden Reihenfolge abhing — einschließlich der Erwartungen in Tests —, muss seine eigene Ordnung herstellen. Das gilt für C++-Code und für die Testsuite ebenso wie für Skripte.

12.4 Invarianten der Registrierung

Die Migration war daran gebunden, dass die Script API nicht abdriftet. Die Invarianten, die konstant gehalten wurden und die auch von künftiger Arbeit in dieser Schicht konstant gehalten werden sollten:

Invariante	Wert	Geltungsbereich
Aufrufstellen <code>register_function(vm</code>	79	<code>src/simutrans/script/</code>
Aufrufstellen <code>log_squirrel_type</code>	7	<code>src/simutrans/script/api/</code>
Abweichung bei der Registrierung	0	—
Abweichung bei der Aufrufkonvention	0	—
Abweichung bei öffentlich verfügbaren Typen	0	—

Die Geltungsbereiche sind ausdrücklich angegeben, weil sie leicht falsch zu treffen sind: Zählt man `register_function(vm` allein unter `src/simutrans/script/api/`, ergibt das 77 und nicht 79. Beide Zahlen sind für ihren jeweiligen Geltungsbereich richtig; nur die weiter gefasste ist die Invariante.

12.5 Das Prinzip der Erhaltung

Die Regel, der diese Migration gefolgt ist und die sich für jede künftige Runtime-Aktualisierung zu bewahren lohnt: **Eine Runtime-Aktualisierung darf ändern, wie die VM angesteuert wird, aber sie darf nicht ändern, was die Script API bereitstellt.** Wenn eine Runtime-Änderung eine sichtbare API-Änderung erzwingt, ist das eine eigene, getrennt geprüfte Arbeit.

13. Stabilität der Script API

Die Migration wurde auf unbeabsichtigtes Abdriften der öffentlichen API geprüft, statt als sicher angenommen zu werden. Die Prüfungen wurden auf einem frischen Checkout des offiziellen Trunk bei r12168 ausgeführt.

Ergebnis: keinerlei Abweichung.

- Anzahl registrierter Funktionen: **79**, unverändert gegenüber dem eingefrorenen Stand vor der Migration (Geltungsbereich: `src/simutrans/script/`).
- Stellen mit Typprotokollierung: **7**, unverändert (Geltungsbereich: `src/simutrans/script/api/`).
- Abweichung bei der Registrierung: **0**.
- Abweichung bei der Aufrufkonvention: **0**.
- Abweichung bei öffentlich verfügbaren Typen: **0**.
- Dateien, die durch die Dokumentationsrevision geändert wurden und Code enthalten: **0**.

Schlicht gesagt: Jede Funktion, die die Script API vor der Migration angeboten hat, wird auch danach angeboten, unter demselben Namen, mit denselben Argumenten und denselben bereitgestellten Typen.

14. Zusammenfassung der Validierung

Alle nachstehenden Zahlen stammen aus einem **frischen Checkout des offiziellen Trunk bei r12168**, gebaut und ausgeführt, nachdem die Commits veröffentlicht waren — nicht aus der Arbeitskopie, aus der die Commits gemacht wurden.

284 / 284

REGRESSION

10 / 10

VM-TESTS

2 / 2

VERWAISTE TESTS

Prüfung	Ergebnis
Vollständige Regressionssuite	284 / 284
VM-spezifische Tests	10 / 10 vorhanden und bestanden
Verwaiste Tests (Tests, die bei einer Regression der Suite stillschweigend verschwinden würden)	2 / 2 vorhanden und bestanden
Neue Squirrel-Fehler	0
Abweichung der Script API	0 (siehe Abschnitt 13)
Dokumentations-Build	269 HTML-Dateien, 0 neue Diagnosen gegenüber der akzeptierten Grundlinie

Zusätzlich während der Migration durchgeführte Validierung:

- Die beiden mitgelieferten skriptgesteuerten KI-Spieler wurden in einer Kampagne gegen die neue Runtime ausgeführt.
- Ein von der bisherigen Runtime geschriebener Spielstand wurde wiederholt geladen, um zu bestätigen, dass das Ergebnis reproduzierbar war und kein einzelner Glücksfall.
- Ein unter 3.2 geladener Zustand wurde erneut gespeichert und erfolgreich neu geladen.
- Beide Zwischenstände der Runtime wurden als unabhängig auslieferbar überprüft, bevor etwas veröffentlicht wurde, sodass keine Revision im Bereich r12165–r12168 den Baum in einem defekten Zustand zurücklässt.

Die Testsuite wird namentlich geprüft statt über die Anzahl: Die VM- und die verwaisten Tests werden *namentlich* als vorhanden bestätigt, sodass eine Suite, die stillschweigend auf eine ältere Liste zurückgefallen ist, erkannt und nicht als bestandene Gesamtzahl gemeldet wird.

15. Bekannte Einschränkungen und Anmerkungen

15.1 Iterationsreihenfolge von Tabellen

Dokumentiert in [Abschnitt 6](#). Dies ist der eine Verhaltensunterschied, den ein vorhandenes Skript beobachten kann, und er ist die Folge einer Upstream-Änderung und keiner Entscheidung von Simutrans.

15.2 Geltungsbereich der Kompatibilitätsaussage

Die Kompatibilitätsaussage deckt die mit Simutrans ausgelieferten skriptgesteuerten KI-Spieler und die in Abschnitt 5 aufgeführten Spielstandrichtungen ab. **Skripte von Dritten wurden nicht getestet.**

15.3 Umgekehrte Richtung nicht zugesichert

Das Laden eines unter Squirrel 3.2 geschriebenen Spielstands in einen älteren Simutrans-Build war nicht Teil dieser Arbeit, und es wird nicht zugesichert, dass es funktioniert.

15.4 Neue Merkmale sind nicht abwärtskompatibel

Die Inline-Form von `bindenv` und `table.map` sind neu. Ein Skript, das eines von beiden verwendet, läuft nicht auf älteren Simutrans-Versionen. Wenn ein Skript auf beiden laufen muss, verwenden Sie `bindenv` als Methode und vermeiden Sie `table.map`.

15.5 Die Script-API-Dokumentation bauen

Diese Anmerkung ist nur von Belang, wenn Sie die Script-API-Referenz selbst mit Doxygen bauen.

`documentation/DoxyfileSQAPl.in` führt `FILTER_PATTERNS = *.cc`, was neuere Doxygen-Versionen zurückweisen. Das ist **vorbestehend und ohne Bezug zur Squirrel-3.2-Runtime** — es war vor dieser Migration vorhanden und wurde bewusst unangetastet gelassen, statt innerhalb einer nicht zusammenhängenden Änderung behoben zu werden. Es verdient einen eigenen kleinen Patch.

Ein zweiter, umgebungsbedingter Punkt für Windows-Erstellende: Der Dokumentations-Build muss den Eingabefilter über einen Shell-Wrapper aufrufen. Ohne ihn beendet sich das native Windows-Doxygen erfolgreich und lässt dabei stillschweigend die gesamte API-Referenz weg. Prüfen Sie die Anzahl der erzeugten Dateien, nicht den Rückgabewert.

16. Wo was zu finden ist

Dieses Dokument verdoppelt die Script-API-Referenz nicht. Verwenden Sie es als Wegweiser.

Wenn Sie Folgendes suchen	Dann hier
API-Details — Klassen, Methoden, Parameter, Konstanten	Die Script-API-Referenz, erzeugt aus <code>src/simutrans/script/api/</code> (<code>api_doc.h</code> ist ihre Hauptseite)
Was die API gewonnen oder geändert hat und wann	<code>src/simutrans/script/api/changelog.h</code> → die Seite <i>Changelog</i>

Wenn Sie Folgendes suchen	Dann hier
Migration und Kompatibilität für Squirrel 3.2	Dieses Dokument sowie <code>src/simutrans/script/api/squirrel_32_en.h</code> → die Seite <i>Squirrel 3.2 runtime update</i> , das Gegenstück im Quelltext
Dasselbe auf Spanisch	<code>src/simutrans/script/api/squirrel_32_es.h</code> → <i>Actualización del runtime de Squirrel 3.2</i>
Häufige Fehler beim Skripten	Die Seite <i>Common pitfalls</i> in <code>api_doc.h</code>
Verfügbare Funktionen der Squirrel-Standardbibliothek	Die Seite <i>Available functions from the squirrel standard lib</i> in <code>api_doc.h</code>
Ausgearbeitete Beispielskripte	Die Beispielseiten in <code>api_doc.h</code>

Die Seite im Quelltext unter `squirrel_32_en.h` ist die **kanonische technische Referenz** für diese Migration. Dieses Dokument ist ihre öffentliche, eigenständige Darstellung: Es ergänzt die zielgruppenspezifische Anleitung in den Abschnitten 7–10, die Validierungszahlen in den Abschnitten 13–14 und die Hinweise zur Einbettung in Abschnitt 12, und es lässt nichts Wesentliches von der Seite im Quelltext aus.

Die spanische Seite in `squirrel_32_es.h` ist eine **geprüfte Übersetzung**; die englische Seite bleibt maßgeblich, wo die beiden je unterschiedlich gelesen werden könnten.

17. Verfügbarkeit und Änderungsprotokoll

Integriert im Trunk von Simutrans Standard: r12165 – r12168, mit r12168 als dem Stand der abgeschlossenen Migration und Dokumentation.

Verfügbar in Builds, die r12168 oder später enthalten. Eine nummerierte stabile Veröffentlichung wird hier nicht zugesichert; prüfen Sie die Veröffentlichungshinweise des Builds, den Sie verwenden.

Änderungsprotokoll

Revision	Änderung
r12165	CODE: update Squirrel instance lookup API for 3.2 — jede Aufrufstelle an die neue Signatur von <code>sq_getinstanceup()</code> anpassen und dabei das bisherige Verhalten erhalten.
r12166	CODE: update Squirrel bindenv support for 3.2 — das Bindungsziel durch die Closure-Erzeugung führen, damit die Inline-Form der Umgebung kompiliert und läuft.
r12167	CODE: reconcile remaining Squirrel 3.2 runtime changes — der Upstream-String-Hash, <code>table.map</code> sowie die verbleibenden Unterschiede in VM und Headern, dazu die Lizenzdatei der Runtime.
r12168	DOC: document Squirrel 3.2 runtime migration — die zweisprachige Migrationsreferenz im Quelltext und ihre Verlinkung von der Hauptseite und dem Änderungsprotokoll.

18. Dokumenthistorie

Version	Datum	Änderung
1.0	2026-08-16	Erste öffentliche Veröffentlichung. Erstellt auf Grundlage der validierten Migration im Trunk bei r12168.