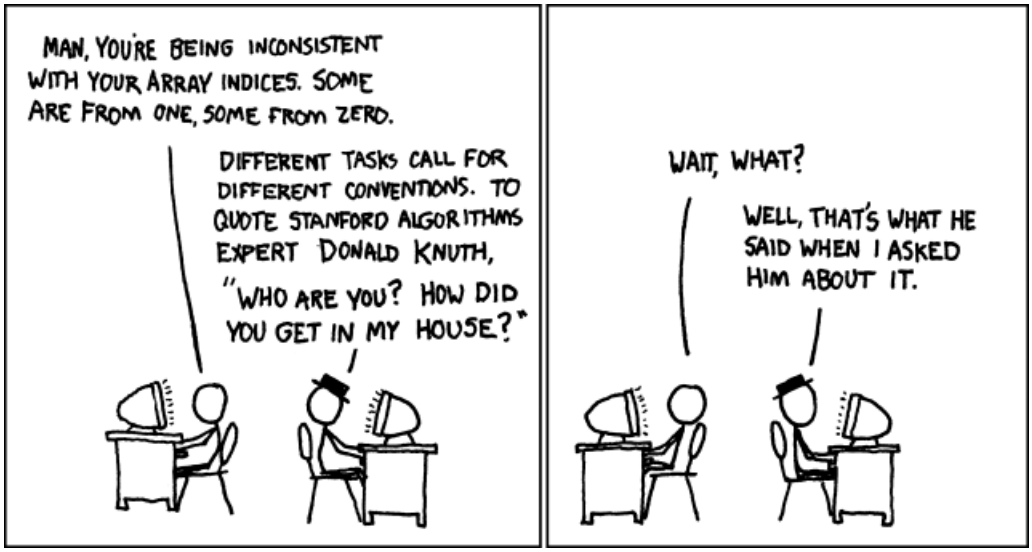
Übung 10: Agenten und Brettspiele

ETH Zurich

Ausgabe: 3. Dezember 2010
Abgabe: 19. Dezember 2010



Donald Knuth © Randall Munroe (xkcd.com)

Ziele

- Ihr Verständnis von Agenten und Ereignisorientierte Programmierung prüfen.
- Ihre Fertigkeit in der Fehlerbeseitigung prüfen.
- Den Entwurf und die Implementation des Brettspiels beenden.

1 Klimaanlage

Sie implementieren die Software für eine Klimaanlage. Nehmen Sie an, dass eine Hardwarekomponente existiert, welche die Temperatur misst und bei einer Änderung der Temperatur eine spezifische Routine der Applikation aufruft.

Zusätzlich haben Sie zwei Klassen bekommen, welche verschiedene Reaktionen auf die Temperaturänderung implementieren. Die Klasse *DISPLAY* repräsentiert ein LED-Display und hat ein Feature *show* (*a_temperature*: *DOUBLE*), dass die Temperatur auf dem Display anzeigt. Die Klasse *HEATING_CONTROLLER* hat ein Feature *adjust* (*a_temperature*: *DOUBLE*), welches

die Heizung und die Kühlung ein- oder ausschaltet, je nachdem wie die Differenz der aktuellen Temperatur zur Zieltemperatur ist.

Ihre Aufgabe ist es die Klasse *TEMPERATURE_SENSOR* zu implementieren. Diese Klasse erhält das Signal der Hardwarekomponente und führt die entsprechenden Aktionen aus.

Aufgabe

1. Erstellen Sie ein neues EiffelStudio-Projekt mit der Wurzelklasse *APPLICATION*. Laden sie die Klassen *DISPLAY* und *HEATING_CONTROLLER* von http://se.ethz.ch/teaching/2010-H/eprog-0001/assignments/10/ac.zip herunter, und fügen Sie diese zum Projekt hinzu.

2. Erstellen Sie eine neue Klasse *TEMPERATURE_SENSOR* mit folgender Funktionalität:

   - Die Klasse sollte die aktuelle Temperatur speichern, und ein Feature *set_temperature* (*a_temperature*: *REAL*) zur Modifikation dieses Werts anbieten. (Dieses Feature wird von der Hardwarekomponente aufgerufen, wenn die Temperatur ändert).
   - Die Klasse sollte eine Möglichkeit anbieten um Agenten als *Beobachter* zu registrieren. Alle diese Beobachter sollten jedes Mal aufgerufen werden, wenn die Temperatur ändert.
   - Es sollte möglich sein, eine beliebige Anzahl dieser Beobachter zu registrieren.
   - Es sollte möglich sein die Features *show* und *adjust*, welche oben beschrieben sind, ohne weitere Änderungen an den Klassen *DISPLAY* und *HEATING_CONTROLLER* zu registrieren.

3. Testen Sie ihre Implementation der Klasse *TEMPERATURE_SENSOR* in der Klasse *APPLICATION*.

   Erstellen Sie dazu Objekte des Typs *TEMPERATURE_SENSOR*, *DISPLAY* und *HEATING_CONTROLLER*, und registrieren Sie die Features *show* und *adjust* beim Sensor. Um die Hardwarekomponente zu simulieren, können Sie *set_temperature* des Sensors mehrmals aufrufen (als Resultat dieser Aufrufe sollte die Temperatur angezeigt und die Heizung angepasst werden).

Abgabe

Reichen Sie den Code der Klasse *TEMPERATURE_SENSOR* und *APPLICATION* ein.

2 Fehlerbeseitigung

In dieser Aufgabe erhalten Sie eine Klasse, die Fehler ("bugs") enthält, sowie ein Test-Szenario, das diese Fehler aufdeckt. Ihre Aufgabe ist es, die Fehler so zu korrigieren, dass das Test-Szenario ohne Fehler (Vertrags-Verletzungen oder Void-Aufrufe) ausgeführt werden kann.

Aufgabe

1. Laden Sie die Datei http://se.ethz.ch/teaching/2010-H/eprog-0001/assignments/10/debugging.zip herunter, entpacken sie in ein Verzeichnis ihrer Wahl und öffnen Sie das Projekt "debugging.ecf" in EiffelStudio.

2. Die Klasse *SORTED_LINKED_LIST* stellt eine einfach verkettete Liste dar, deren Elemente sortiert sind. Die Implementierung der Klasse enthält 10 Fehler[1]. Die Wurzelklasse *TESTER* enthält eine Test-Routine, die die Features von *SORTED_LINKED_LIST* aufruft. Jeder der 10 Fehler wird durch die Test-Routine aufgedeckt indem entweder die Verträge von *SORTED_LINKED_LIST* oder **check**s in der Routine selbst verletzt werden, oder ein Void-Aufruf verursacht wird.

3. Führen Sie das Programm aus: dies wird in einem Fehler resultieren. **Verwenden Sie den Debugger** (schrittweise Ausführung, "Objects"-Werkzeug, "Watch"-Werkzeug) um die Ursache des Fehlers herauszufinden, und korrigieren Sie die Implementierung von *SORTED_LINKED_LIST*, so dass er nicht mehr auftritt. Hinweis: Verändern Sie nur Feature-Rümpfe in *SORTED_LINKED_LIST*; Sie dürfen keine Verträge verändern, Features hinzufügen oder entfernen, oder die Klasse *TESTER* verändern.

   Wiederholen Sie diese Schritte, bis das Programm ohne Fehler ausgeführt werden kann.

Abgabe

Geben Sie die von Ihnen modifizierte Klasse *SORTED_LINKED_LIST* und eine Beschreibung der korrigierten Fehler (jeweils ein Satz) ab. (Die Beschreibung der Fehler kann entweder in einem separaten Dokument erfolgen, oder als Kommentare im Programmtext.)

3 Das Abschlussprojekt: Brettspiel – Teil 4

Für das Abschlussprojekt können Sie eine der nachfolgenden Optionen wählen:

1. Vervollständigen Sie die Implementation des Brettspiels gemäss der unten angegeben Spezifikation.

2. Implementieren Sie ein Spiel Ihrer Wahl, vorausgesetzt Ihr Assistent hält das Projekt für angemessen.

Wenn Sie Option 1 gewählt haben, dann müssen Sie die folgende vereinfachte Version von *Monopoly* implementieren (http://en.wikipedia.org/wiki/Monopoly_(game)), welche nach folgenden Regeln gespielt wird.

Das Spiel besteht aus einem Spielbrett (Abbildung 1) mit 20 Feldern, zwei 4-seitigen Würfeln (jeweils Augen 1 bis 4) und kann von 2–6 Spielern gespielt werden.

Der Spielablauf ist folgendermassen:

- Spieler haben Geld und können Grundstücke besitzen. Jeder Spieler startet mit 1500 CHF, aber ohne Grundstücke.

- Alle Spieler starten auf dem ersten Feld ("Go").

- Ist ein Spieler am Zug: rollt er den Würfel und setzt seine Spielfigur entsprechend der Augenzahl im Uhrzeigersinn weiter. Wenn das 20. Feld erreicht wurde, wird die Spielfigur wieder auf das erste Feld gesetzt.

- Bestimmte Felder beeinflussen den Spieler (siehe nachfolgend), wenn seine Spielfigur das Feld passiert oder darauf steht. Insbesondere kann die Geldmenge des Spielers beeinflusst werden.

- Wenn ein Spieler nach seinem Zug einen negativen Geldbetrag hat, dann scheidet er aus dem Spiel aus. Alle seine Eigentümer werden herrenlos.

[1] Die Anzahl der Fehler ist nur ein Näherungswert; die exakte Anzahl hängt von der Definition von "Fehler" ab.

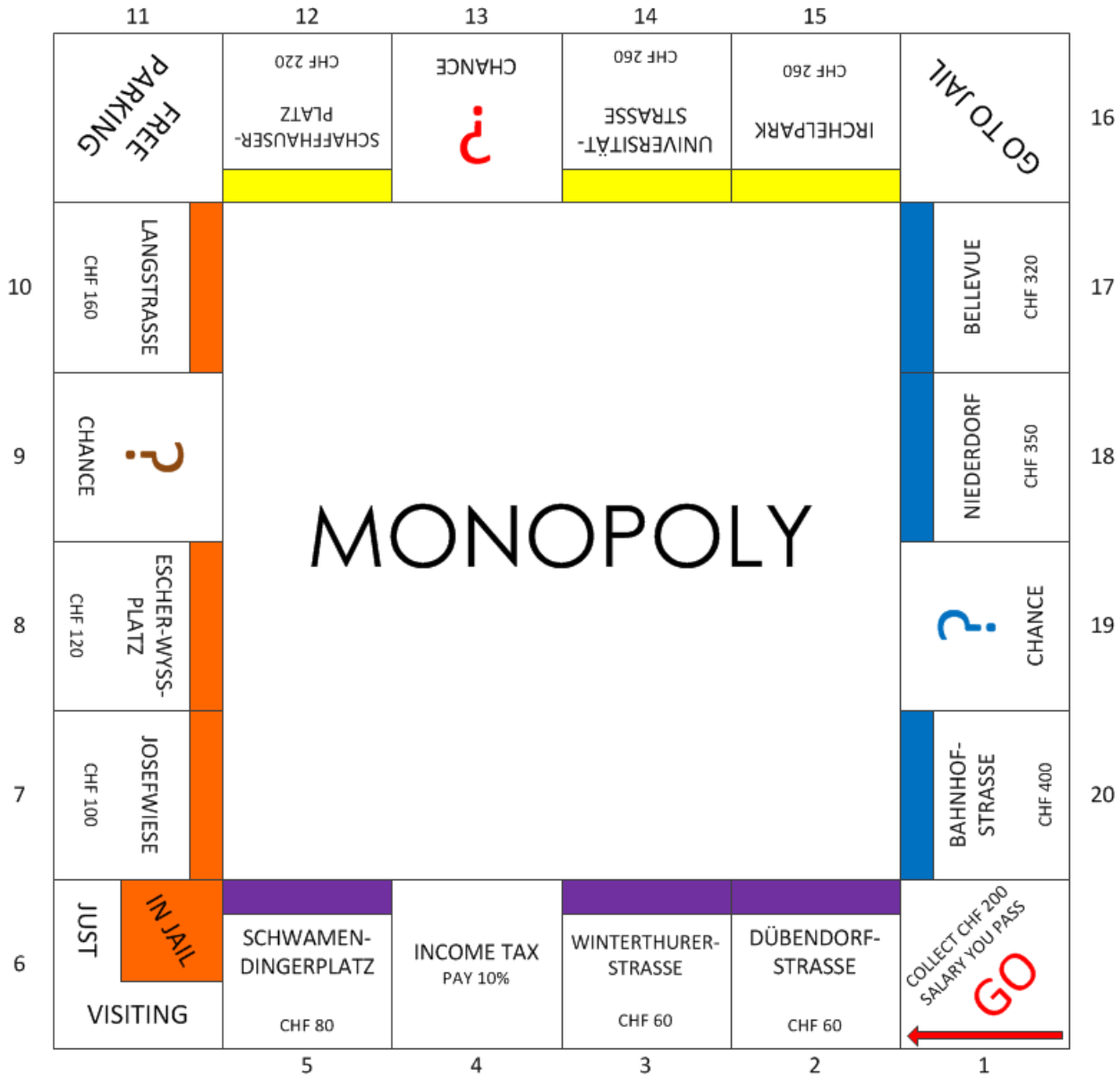


Abbildung 1: Monopoly Spielbrett

- Eine Runde endet wenn jeder Spieler einmal einen Zug getan hat.
- Das Spiel endet entweder wenn nur noch ein einziger Spieler übrig bleibt oder wenn 100 Runden gespielt wurden. Der Gewinner ist der Spieler mit dem meisten Geld. Ein Unentschieden (mehrere Gewinner) ist möglich.

Es gibt folgende Felder auf dem Spielbrett:

Grundstücksfelder (gekennzeichnet durch einen farbigen Streifen). Sie enthalten den Namen und den Preis eines Grundstücks und können von einem Spieler in besessen werden. Wenn ein Spieler auf einem nicht-besessenem Grundstück landet, hat er die Option dieses für

den angegeben Preis zu kaufen. Wenn ein Spieler auf einem Grundstücksfeld landet, dass von einem anderen Spieler besessen wird, dann muss er dem Besitzer Miete zahlen (der Betrag ist in Tabelle 1 angegeben).

Go. Jedesmal wenn ein Spieler dieses Feld passiert (er muss nicht direkt darauf landen), bekommt er 200 CHF ausgezahlt.

Chance. Wenn ein Spieler auf einem solchen Feld landet, dann bekommt er entweder einen zufälligen Geldbetrag (ein Vielfaches von 10) jedoch maximal 200 CHF, oder er verliert einen zufälligen Geldbetrag (ein Vielfaches von 10) jedoch maximal 300 CHF.

Income tax. Wenn ein Spieler auf diesem Feld landet, zahlt er 10% seines Geldes (abgerundet auf ein Vielfaches von 10) als Steuer.

Free parking. Dieses Feld hat keine Auswirkungen.

Go to Jail. Wenn ein Spieler auf diesem Feld landet, dann kommt er direkt in den "In Jail" Teil des "In Jail/Just Visiting" Feldes.

In Jail/Just Visiting. Wenn ein Spieler auf diesem Feld landet, dann ist er nur ein Besucher ("Just Visiting" des Feldes) und es gibt keine weitere Auswirkung.

Wenn ein Spieler auf dieses Feld kam, weil er auf dem Feld "Go to Jail" gelandet ist, dann sitzt er im Gefängnis und kann seine Spielfigur nicht einfach weitersetzen. Ist ein Spieler am Zug, der sich im Gefängnis befindet, wird er gefragt ob er 50 CHF zahlen möchte um freizukommen. Alternativ kann der Spieler 3 Runden lang versuchen einen Pasch[2] zu würfeln. Bei einem Pasch kommt der Spieler frei. Wurde nach der dritten Runde noch kein Pasch gewürfelt, muss der Spieler die Strafgebühr zahlen. Wenn der Spieler freikommt (durch Pasch oder Strafgebühr) setzt er sofort seine Spielfigur gemäss der gewürfelten Augenzahl weiter.

Tabelle 1: Grundstücke

Position	Name	Price	Rent
2	Dübendorfstrasse	60	2
3	Winterthurerstrasse	60	4
5	Schwamendingerplatz	80	4
7	Josefwiese	100	6
8	Escher-Wyss-Platz	120	8
10	Langstrasse	160	12
12	Schaffhauserplatz	220	18
14	Universitätstrasse	260	22
15	Irchelpark	260	22
17	Bellevue	320	28
18	Niederdorf	350	35
20	Bahnhofstrasse	400	50

Aufgabe

Implementieren Sie das Spiel mit einem Kommandozeilen-Interface. Ihr Programm soll jedes Mal eine Eingabe erwarten, wenn ein Spieler ein Grundstück kaufen kann oder eine Strafzahlung vornehmen kann um aus dem Gefängnis zu gelangen.

[2]Beide Würfel zeigen dieselbe Augenzahl.

Wir empfehlen, dass Sie mit der Musterlösung von Übung 8 beginnen:
http://se.ethz.ch/teaching/2010-H/eprog-0001/assignments/08/board_game_solution.zip

Nach Wunsch könnnen Sie beliebige Erweiterungen des Spiels vornehmen, zum Beispiel:

- weitere Standard-Regeln von Monopoly hinzufügen: Grundstücks-Gruppen, Auktionen, Grundstücke bebauen, Beleihung, usw.
  (siehe z.B. http://richard_wilding.tripod.com/monorules.htm);
- Computer Gegner (Ihr Programm spielt einige der Gegner);
- eine graphische Anwenderschnittstelle.

Abgabe

Geben Sie die Ihren Programmcode ab.