MODEL BASED TEST DESIGN AT UNITY

Marek Turski, Ilya Turshatov, Tomasz Paszek

Unity Technologies
Unity Technologies

Provider of an integrated development environment for creating games and other interactive virtual content
Unity Engine

- A few statistics
  - 100 core product developers
  - 1 million monthly active developers
  - 45k unique titles made with Unity every month
  - 9 million new devices reached every day
  - 20 million new install events per day
Unity Engine

- Scaling is challenging
  - Growing team size
  - 23 supported platforms
  - Integrated services
    - Player retention
    - Cloud deployment
    - Multiplayer
    - Asset Store
    - Analytics
    - Ads
QA Focus

• Manual Testing (Test Engineers)
• User Experience (UX Researchers)
• User Support
  • Student Workers
  • Support Engineers
QA Focus

• Test Automation (Test Developers)

• Test Infrastructure (Toolsmiths)
  • Test runners and frameworks
  • Bug reporting and customer support tools
  • Backend and Reporting
  • Working closely with the Build Infrastructure Team
Test Automation Focus

• Unit Tests
  • Code tests, written by developers

• System Integration Tests
  • Components seen as processes

• Sub-System Runtime Tests
  • Components seen as interfaces

• Test Tools
  • Built directly into the Game Engine
Test Automation Focus

- Non-Functional Tests
  - Performance/Stress/Load
  - Deployment/Update/Security
  - User Interface
  - Asset Import
  - Graphics
Towards Agile Development

Individuals and interactions over Processes and tools
Working software over Comprehensive documentation
Customer collaboration over Contract negotiation
Responding to change over Following a plan

Is Unity agile (enough)?
Towards Agile Testing

• Re-evaluation of skills
  • Planning
  • Communication
  • Quality Assistance over Quality Assurance

• Toolbox Clean-up
  • Concepts
  • Techniques
  • Testware
Good Requirements Specification

- Traditional guidelines
  - Clarity, Completeness, Correctness, Consistency
- Agile User Stories
  - Small and Independent
  - Estimable and Testable
  - Valuable and Negotiable

As a [user role] I want to [desired feature] so that [value/benefit].
“Start programming while I find out what they want...”
Model-Based Requirements

• Model-Based Testing
  • Derivation of test cases from a model of the desired system behaviour

• Why use models?
  • The modelling process improves our understanding of the system under test and finds inconsistencies earlier
  • Models become collaborative (and negotiable) test artifacts which visualize and document our view of the system
Model-Based Requirements

• Properties of a good model-based notation
  • Expressive
    • Represents common process abstractions and control issues
  • Provocative
    • Helps with or drives the discovery of system aspects
  • Processable
    • Can be data-mined or executed
  • Scalable
    • Provides means of functional or data decomposition
Model-Based Requirements

Expressing Common Behaviours

- Sequence
- Structured Control
  - Repetition
  - Selection
  - Activate
  - Enable / Disable
  - Trigger / Suspend
  - Pause / Resume

- Events
  - Asynchronous events
  - Context-sensitive input events
  - Multi-context output events
  - Event quiescence

- Tasks
  - Priority
  - Parallel execution
  - Mutual exclusion

- Simultaneous Tasks

Deadlock
Modelling System Requirements

Views of a System

Structural

Behavioural

Control Flow
Data Flow
State Transitions

Structural modelling focuses on what the system is.
Behavioural modelling focuses on what the system does.
Control Flow Modelling

- **Behavioural**
  - **Control Flow**
  - **Data Flow**
  - **State Transitions**

<table>
<thead>
<tr>
<th>Emphasizes</th>
<th>The exact sequence of steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>De-emphasizes</td>
<td>How inputs to a state are determined</td>
</tr>
<tr>
<td><strong>Flow</strong></td>
<td>Control stream - the next step is taken when a previous step (or sequence of steps) has finished</td>
</tr>
</tbody>
</table>
Flowcharts

- One of the earliest behavioural models
- Illustrates algorithms, processes and workflows
- Multitude of styles and symbols

<table>
<thead>
<tr>
<th>Emphasizes</th>
<th>The exact sequence of steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>De-emphasizes</td>
<td>How inputs to a state are determined</td>
</tr>
<tr>
<td>Flow</td>
<td>Control stream - the next step is taken when a previous step (or sequence of steps) has finished</td>
</tr>
</tbody>
</table>
Flowcharts

• “(...) algorithms, processes and workflows” - what is the difference?
Flowcharts

• “(...) algorithms, processes and workflows” - what is the difference?

Referee

Divide into teams

Pick a card

Next player

Next team

Yes

Picture it

Can they guess it?

Team gets a point

No
Leap Year Problem flowchart

static bool IsLeapYear(int year) {
    return (year % 4) == 0 && ((year % 100) != 0 || (year % 400) == 0);
}
Coffee Machine Problem flowchart

Insert coin

Coin is?

2
5
10

Yes

No

total = 0

total += 2

total += 5

total += 10

Dispense coffee, return change

total >= 20?
Flowcharts – Decomposition

• Layout Decomposition
  • On-page connectors
  • Off-page connectors

• Hierarchy Decomposition

Predefined Process

Compute leapYear

Leap Year?

False

True

Off-page connector
Flowcharts – Getting Started

• How to start?
  • Define the process to be illustrated
  • Find a trigger event or a trigger input
  • List known actions, try to keep the order chronological

• Extending the model
  • What could/should happen next?
  • Are all important aspects of the process illustrated?
  • Are all relevant actions taken into account?
Days in a Month flowchart

1. day, month, year
2. ?
Days in a Month flowchart

day, month, year

February?

Yes

Leap Year?

False

28

True

29

No

Leap Year?
Days in a Month flowchart

- day, month, year
- February?
  - Yes
  - Leap Year?
    - True
      - 29
  - No
    - False
    - 28
- Yes
  - Month 4, 6, 9 or 11?
    - Yes
      - 30
    - No
      - 31
Turn-based game flowchart

Start Turn

Highligh Available Locations

Select Location

Move Character

Move

Action?

Attack

Highlight Available Opponents

Select Opponent

Attack Animation

Show Available Spells

Select Spell

Yes

Opponent Adjacent?

Player Eliminated?

Yes

Defeat Animation

No

Return Attack Animation

Yes

Opponent Eliminated?

End Turn

No

Victory Animation
Flowcharts – Processability

• Well formed flowcharts can be data-mined by business workflow engines

• An executable variation of flowcharts is used by the DRAKON Visual Language
  • Automatic layout
    • Vertical only
    • Unified distances and offsets
    • Logical alignment
  • Silhouette design style
  • Joins and parallel actions
Flowcharts – Expressive power

Structured Control Statements are the core building blocks of a flowchart

- Perfect for representing imperative languages

![Diagram of Sequence, Selection, and Repetition flowchart elements]

- Sequence
- Selection
- Repetition
Flowcharts – Expressive power

- Other aspects of behaviour must be determined in text or deducted from the flow
  - Mutual exclusion is represented through parallel paths after a decision, but other concurrency concepts are missing
  - A popular extension - Activity Diagrams – addresses this problem by introducing new symbols

- Join
- Fork
- Merge
- Branch
Break Time

- 5 minute break
- Q&A
- Flowchart design exercise (optional)
State Transition Modelling

- **Behavioural**
  - Control Flow
  - Data Flow
  - State Transitions

<table>
<thead>
<tr>
<th>Emphasizes</th>
<th>The state space and state transition stimuli</th>
</tr>
</thead>
<tbody>
<tr>
<td>De-emphasizes</td>
<td>Sequencing of steps</td>
</tr>
<tr>
<td>Flow</td>
<td>Event stream – the next step is taken in response to events and the internal state of the system</td>
</tr>
</tbody>
</table>
Finite State Machines

- A mathematical model of computation (a finite automata)
- Illustrate behaviour as a series of events that can occur in one or more possible states of a system

<table>
<thead>
<tr>
<th>Emphasizes</th>
<th>The state space and state transition stimuli</th>
</tr>
</thead>
<tbody>
<tr>
<td>De-emphasizes</td>
<td>Sequencing of steps</td>
</tr>
<tr>
<td>Flow</td>
<td>Event stream – the next step is taken in response to events and the internal state of the system</td>
</tr>
</tbody>
</table>
Common Finite State Machines

Tick

S1 Green

Change

S2 Yellow

Tick

S3 Red

Change

S1 Patrol

Respawn

S3 Base

Tick

Pac-Man in sight

S2 Chase

Lost Pac Man

Power Pill

Power Pill wears off

S4 Retreat

Pac Man catches up

User Conference on Advanced Automated Testing
Leap Year Problem FSM

```csharp
class Year
{
    bool m_IsLeapYear;

    public Year(int year)
    {
        m_IsLeapYear = IsLeapYear(year);
    }
}
```

S1 Get year

S2 Divisible by 4?

S4 Divisible by 100?

S6 True

S3 False

S5 Divisible by 400?

Yes

Yes

No

Yes

No

No
Day of Month Problem FSM

S1 February?
- No
  - S7 Month 4, 6, 9 or 11?
    - No
      - S9 31
      - No
    - Yes
      - S4 Year divisible by 100?
        - No
          - S6 28
          - No
        - Yes
          - S5 Year divisible by 400?
            - Yes
              - S2 Year divisible by 4?
                - Yes
                  - S3 29
                - No
          - S5 Year divisible by 400?
            - No
              - S3 29
            - Yes
              - S4 Year divisible by 100?
                - No
                  - S6 28
                  - No
                - Yes
                  - S5 Year divisible by 400?
                    - Yes
                      - S2 Year divisible by 4?
                        - Yes
                          - S3 29
                        - No
                      - S2 Year divisible by 4?
                        - No
                          - S3 29

S8 30
- Yes
  - S7 Month 4, 6, 9 or 11?
    - No
      - S9 31
      - No
    - Yes
      - S4 Year divisible by 100?
        - No
          - S6 28
          - No
        - Yes
          - S5 Year divisible by 400?
            - Yes
              - S2 Year divisible by 4?
                - Yes
                  - S3 29
                - No
          - S5 Year divisible by 400?
            - No
              - S3 29
            - Yes
              - S4 Year divisible by 100?
                - No
                  - S6 28
                  - No
                - Yes
                  - S5 Year divisible by 400?
                    - Yes
                      - S2 Year divisible by 4?
                        - Yes
                          - S3 29
                        - No
                      - S2 Year divisible by 4?
                        - No
                          - S3 29

S2 Year divisible by 4?
- Yes
  - S3 29
- No
  - S3 29

S4 Year divisible by 100?
- Yes
  - S5 Year divisible by 400?
    - Yes
      - S2 Year divisible by 4?
        - Yes
          - S3 29
        - No
      - S2 Year divisible by 4?
        - No
          - S3 29
    - No
      - S2 Year divisible by 4?
        - Yes
          - S3 29
        - No
      - S2 Year divisible by 4?
        - No
          - S3 29

S3 29
- Yes
  - S2 Year divisible by 4?
    - Yes
      - S3 29
    - No
      - S3 29
- No
  - S3 29
Day of Month Problem FSM

- **S1**: February?
  - Yes: Leap Year?
  - No: Month 4, 6, 9 or 11?

- **S2**: Leap Year?
  - Yes: S3 (29)
  - No: S4 (28)

- **S3**: 29

- **S4**: 28

- **S5**: Month 4, 6, 9 or 11?
  - Yes: S6 (30)
  - No: S7 (31)

- **S6**: 30

- **S7**: 31
Coffee Machine Problem FSM

S1: Total is 0
S2: Total is 10
S3: Total is 5
S4: Total is 15
S5: Total is 20

- **S1** to **S2**: Insert 5
- **S2** to **S3**: Insert 5
- **S2** to **S4**: Insert 10
- **S2** to **S5**: Insert 10
- **S3** to **S5**: Insert 5
- **S4** to **S5**: Insert 5
- **S5** to **S1**: Take Coffee
Data Flow Modelling

- **Behavioural**
  - **Control Flow**
  - **Data Flow**
  - **State Transitions**

<table>
<thead>
<tr>
<th>Emphasizes</th>
<th>Exchange and transformation of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>De-emphasizes</td>
<td>Sequencing of steps</td>
</tr>
<tr>
<td><strong>Flow</strong></td>
<td>Data stream - the next step is taken when other steps provide its inputs</td>
</tr>
</tbody>
</table>
Petri nets

- Graphical notation for step-wise process analysis
- A mathematical modelling language for the description of distributed systems supported by a well developed theory for process analysis

<table>
<thead>
<tr>
<th>Emphasizes</th>
<th>Exchange and transformation of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>De-emphasizes</td>
<td>Sequencing of steps</td>
</tr>
<tr>
<td>Flow</td>
<td>Data stream - the next step is taken when other steps provide its inputs</td>
</tr>
</tbody>
</table>

*User Conference on Advanced Automated Testing*
Petri Net - Coffee Machine Problem

Looks familiar?

```
P1 Total is 0
P2 Total is 10
P3 Total is 5
P4 Total is 15
P5 Total is 20

Insert 5
Insert 5
Insert 5
Insert 10
Insert 5

Take Coffee
```

Insert 10
Insert 10
Insert 5

Petri Net – Traffic Light Model

Red → Green → Orange → T1 → T2 → T3

Traffic Light Image
Petri Net - Traffic Light Model

Red

T1

Green

T2

Orange

T3
Petri Net – Traffic Light Model

Red → Green → Orange → T1 → T2 → T3
Petri Net – Traffic Light Model

Red

T1

Green

T2

Orange

T3

© All rights reserved
Petri Net – Two Traffic Light Model

R1 R2
T2a
G2
T2b
O2
T2c

R1
G1
T1b
O1
T1c

T1

R2

User Conference on Advanced Automated Testing
Petri Net – Two Traffic Light Model

R1 → T1 → G1 → T1b → O1 → T1c

R2 → T2a → G2 → T2b → O2 → T2c

Traffic Lights:
- R1: Green
- R2: Red

Diagram:
- Places: R1, R2, T1, T2a, T1b, T2b, O1, O2, T1c, T2c
- Transitions: G1, G2

User Conference on Advanced Automated Testing
Petri Net – Two Traffic Light Model

R1 – T1 – G1 – T1b – O1 – T1c – R2

T2a – G2 – T2b – O2 – T2c – R2

Traffic Light:
- Red
- Yellow
- Green
Petri Net – Two Traffic Light Model
Petri Net – Two Traffic Light Model
Petri Net – Two Traffic Light Model
Petri Net – Two Traffic Light Model
Petri Nets – Expressive Power

Events/actions sequence

Non-deterministic events - conflict, choice or decision

Concurrent executions
Petri Nets – Expressive Power

Synchronization and Concurrency
Summary – Model Notations

Flow Chart

Petri Nets

State Machines
## Summary – Model Expressive Power

<table>
<thead>
<tr>
<th>Feature</th>
<th>Flowcharts</th>
<th>Finite State Machines</th>
<th>Petri Nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td>Yes</td>
<td>Yes (unnatural)</td>
<td>Yes (unnatural)</td>
</tr>
<tr>
<td>Selection</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Repetition</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Activate</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Enable/Disable</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Trigger/Suspend</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Pause/Resume</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Priority</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Parallel execution</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Mutual execution</td>
<td>No*</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Deadlock</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Context-sensitive input events</td>
<td>No</td>
<td>Yes</td>
<td>No*</td>
</tr>
<tr>
<td>Multi-context output events</td>
<td>No</td>
<td>Yes</td>
<td>No*</td>
</tr>
<tr>
<td>Asynchronous Events</td>
<td>No</td>
<td>No</td>
<td>No*</td>
</tr>
<tr>
<td>Event quiescence</td>
<td>No</td>
<td>No</td>
<td>No*</td>
</tr>
</tbody>
</table>
Break Time

• 5 minute break
• Q&A
• Installing and setting up Unity (optional)
From Models to Test Tools

The built-in scripting engine makes Unity easy to extend with custom tools, including test tools

- Integrated NUnit and Integration Test Frameworks
- Model Based Test Designer
  - Uses the open-source GraphWalker tool at the core (ver. 3.3)
  - Extended Finite State Machine
  - Intuitive workflow and notation
- Integrated test designer interface
- Builds offline tests that can be executed using the other frameworks
From Models to Test Tools

• **Actions and Guards**
  • Executable annotations that allow for flexible control over path traversals through the model

• **Shared State**
  • Represents the same system state in multiple models (contexts of the same model)
  • The abstraction allows for hyperlink-like jumps between models, where every next model can extend the represented behaviour’s scope or detail level
From Models to Test Tools

Model notation
- **Model**
  - Start State
  - Start Actions
  - Requirements
- **State**
  - Shared State Key
  - Requirements List
- **Transition**
  - Weight
  - Guard
  - Actions
  - Requirements

Model M <start state> {action1; action2; ...} : “requirement1”, “requirement2”, ...

State C
- Transition: T_{BA1}
- Transition: T_{BA2}

State B
- Shared state key: “req1”, “req2”, ...
- Transition: T_{AB} : weight [guard] {action1; action2; ...} : “req1”, “req2”, ...

State A
- Transition: T_{BA1}

State D
From Models to Test Tools

Model notation
• Model
  • Start Actions
• Transition
  • Guard
  • Actions

Model notation

Model M <State A> {var visited = false}

Make ABC illegal - always visit D before C.

State C

T_{BC} [visited]
{visited = false}

State B

T_{BD} {visited = true}

State A

State D

Make ABC illegal - always visit D before C.

Model M <State A> {var visited = false}
Model notation
- Model
- Start Actions
- Transition
- Guard
- Actions

State C

T\_BC [visited]
{visited = false}

T\_BA1 [visited], T\_BA2

State A

State B

T\_BD {visited = true}

State D

Also, make T\_BA1 feedback illegal between D and C.

Model M <State A> {var visited = false}
Model notation

- Model
- **Start Actions**
- Transition
- **Guard**
- Actions

```
Model M <State A> {var visited = false; var round = 0}
```

Also, add an initially disabled $T_{BE}$.

```
State E  
```

```
T_{BE} [round < 5]
{visited = false; round++}
```

```
T_{BC} [visited]
{visited = false; round++}
```

```
T_{BA1} [visited], T_{BA2}
```

```
T_{BD} {visited = true}
```

```
State D
```

```
State B
```

```
State C
```

```
State A
```

```
Also, add an initially disabled $T_{BE}$.
```

From Models to Test Tools

Model notation
- Transition
- Weight

Model M <State B>

Probability \( P(T_{BD}) = \frac{50}{(80 + 50 + 10 + 15 + 1)} = 32\% \)
Model notation

- State
- **Shared State Key**

Consider state **E** when in state **B** (and **B** when in **E**).

Model M <State B>
From Models to Test Tools

Path generators
- Random
- Quick Random
- Weighted
- A Star

What’s the next step?

Model M <State B>
Stop Conditions
- Coverage
  - States
  - Transitions
  - Requirements
- Reached Target
  - (Shared) State
  - Transition
  - Assertion
- Time Duration
- Path Length
- Composite
  - AND/OR Tree

Pick at Random until
Path Length = 4 AND Reached “State D”

Model M <State B>
Let’s see it in action!
Thank You!

Q & A