# Journey into the DAG: Puzzle Dependency Charts, Tentacles and You

**Joshua Weinberg**
The Website is Down

GAME DEVELOPERS CONFERENCE® March 14–18, 2016 · Expo: March 16–18, 2016 #GDC16

UBM

# Intro

- Talk is about Puzzle Dependency Graphs
- I Applied this technique to analyze the game “Day of the Tentacle”
- I discovered non-obvious insights into the design of the game.

# Poster

Journey Into The DAG:

## Puzzle Dependency Charts, Tentacles and You

A Puzzle dependency chart is a Directed Acyclic Graph (DAG).

Nodes represent puzzles. Edges indicate dependencies.

Tools:


Graphviz


Gephi

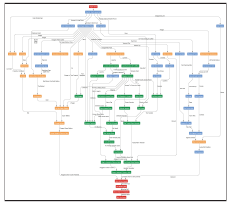

Tulip


yEd

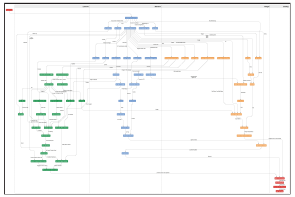

Cambridge Advanced Modeller

Visualizations:

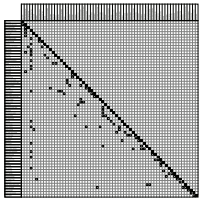
1. Hierarchical Dependency Chart, ungrouped



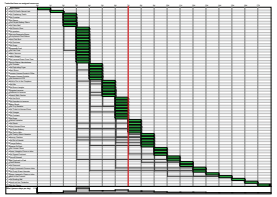
2. Hierarchical Dependency Chart, swimlanes based on character



3. Dependency Structure Matrix



4. Gantt Chart



5. The Day of The Tentacle (LucasArts, 1993) Puzzle Dependency Chart

Hierarchical Layout grouped by downstream goal.
Nodes colored by game character. 63 puzzles in 5 groups.




To download these graphs and the Day of The Tentacle Puzzle Dependency Graph dataset visit:

http://thewebsiteisdown.com/devblog

# In the Talk

- I'll make some bold sweeping statements:
  - Like for example I will define what a puzzle is!
- I'll make some dramatic claims along the way:
  - Like: No two players have ever solved this game in the same way.
  - And: Narrative development is incompatible with non-linear branching.
- Will introduce some practical tools which give layout and analysis advantages.

# Who am I?

- Writer / Director of comedy videos
- The Website is Down (websiteisdown.com)
- Creating an adventure game from the Webseries
- Research for that game led to this analysis

# Puzzle Dependency Graphs What?

- The idea is: Make a diagram with all the puzzles in your game to show how they relate.
- Key insight here is that solving puzzles gives you things you need to solve other puzzles.
- Thus a dependent relationship between puzzles.

# Puzzle Dependency Graphs - Why

- Visualize the puzzles in your game
- Give insight into size, complexity, difficulty and pace of the game.
- Visualizes non-linear Branching
- Not used during actual play, just design.
- Noah Falstein GDC 2013 talk. Credits Ron Gilbert for first use in game design.

# PDG theory: graphs

- A Puzzle Dependency Graph is a type of Dependency Graph. A dependency graph is a type of graph.
- Graph is a general data structure. A collection of nodes and relationships between them.
- Graphs in general can be:
    - Directed / Undirected
    - Connected / Disconnected
    - Cyclic / Acyclic
- In degree and out degree

# PDG theory: Dependency Graph

- Dependency Graph is a special application of a graph in which edges represent dependent relationships between nodes.
- Dependency over time.
- DG have been around since the 60s : PERTT charts
- A common data structure

# DG is a DAG

- A Dependency Graph is
  - Directed
  - Acyclic
- Known as a DAG
- Can be connected or not

# Puzzle Dependency Graph

- PDG is the application of a dependency graph to puzzles in a game.
- Nodes in a PDG represent puzzles.
- Edges represent dependent relationships or “the thing you get from solving the puzzle which you use to solve another puzzle”.

# Puzzle / Dependency Definition

- What is a puzzle?
- *For the purposes of this discussion… A puzzle is anything you do in the game which satisfies the dependencies of other downstream puzzles.*
- What is a dependency?
- *Anything which you have to do to solve a puzzle!*
- Recursive and self-referential definition which is totally lacking in mathematical formalism but still super useful.

# Example

- Need a sword to kill a dragon.
- Sword is in a locked chest.
- Need key to open the chest.
- “Kill the Dragon Puzzle” has a dependency on “Open the Chest puzzle”
- “Open the Chest Puzzle” has a dependency on “Get the Key Puzzle”

# Let's talk non-linear branching

- Our example game is linear - each puzzle has a single dependency and you must solve them each in order.

- Say the Opening the Chest puzzle had two dependencies - The hinge is rusty and you need an oil can to oil it before it will open - then you have a non-linear game segment.

- Player can solve Get the Key puzzle or Get the Oilcan puzzle in any order. Both must be solved to open the chest but the order is not important.

# Day of the Tentacle

- We've covered PDG in general.
- We talked about non-linear branching.
- Let's talk about Day of the Tentacle.
- Classic LucasArts game released in 1993
- Written by Ron Gilbert and he used a PDG to do it.

# The Day of the Tentacle

- Play three characters in a time travel scenario.
- One primary goal - save the world
- Three sub-goals / sub-plots
    - Power Hoagie's Chron-o-John
    - Power Laverne's Chron-o-John
    - Get a Diamond
- PDG is recreated here. Start with #1.

# Day of the Tentacle PDG Features

- Nodes are colored by the character.
- Dependencies show the “thing” you get.
- Graphs all have identical data, just different layouts.

# What can we see here?

- Single starting node
- Single leaf node
- What if multiple starting or ending conditions? This would mean multiple root or leaf nodes.
- Linear story game would have no branching and graph would be just a line.

# Non-Linear Branching in DOTT

- Non Linear Branching is when you have multiple puzzles “in flight” at the same time.
- With all their dependencies satisfied they are ready for the player to solve them.
- For example once you solve **Get Laverne Outside** you can solve
  - Get Dentures to Laverne
  - Get Box o’Laughs
  - Disguise Laverne
- Visualized on the graph by **Layers**

# What's so great about non-linear branching?

- Pros: This has the benefit of giving plenty of things to do at any time and avoids holding up the game because the player is stuck on a single puzzle.

- Cons: (Dramatic claim #2) **Non-Linear Branching makes narrative difficult.**

# Why does Non-Linear Branching make Narrative Difficult?

- Need to talk about **Topological Orders**
- A Topological Order is an order in which you can visit the nodes in the graph such that their dependencies have been resolved by the time you get there.
- In the case of a PDG it is an order in which you can solve the puzzles in the game.
- **Topological Sort** is an algorithm which can find a topological order. Generating a topological order is the most common use for a dependency graph.

# How Many Topological Orders

- There can be more than one valid order for a graph!
- How many?
- To get a lower bound on the answer we can just look at a single layer. Each layer can be fulfilled independently. So pick the third layer. There are 13 puzzles.
- To compute permutations you take factorial of the number of items in the set.
- 13! is > 6 billion ways in which you can arrange the elements of that set. So 6 billion orders those puzzles can be solved.
- Based on this number I stake my dramatic claim #1 (TM) t**hat outside of using a walkthrough no two players have ever solved all the puzzles in this game in the same order.**

# Dynamic Storytelling

- Professor Brian Moriarty's talk about interactive storytelling: I sing the Story Electric. Talk given at Practice 2015.
- A story is **a particular causally related sequence of events**
- Are these 6 billion paths through the graph all different stories?
- They are all particular sequences of events.
- But are these events causally related? We don't know from looking at the graph. But from playing the game we know they are not.

# No narrative thread

- For example there is no narrative thread connecting
  - Get the lab coat
  - Get the Soap
  - Make Vinegar
  - Get the Dentures
- I got the lab coat, then I got the soap, then I made the vinegar, then I got the Dentures.
- These are puzzles with no connection to narrative.

# Narrative Threads Go Vertically

- There is a narrative thread moving down the graph.
    - Get the Soap
    - Start Storm
    - Get kite
    - Charge Battery
    - Power-Hoagie's Chron-o-John
- These events ARE causally related and thus are a story.
- "I got the soap which I used to wash the buggy which caused a rainstorm which caused Ben

# Implications for Narrative

- Non-linear branching makes narrative challenging. You can have both easily but you can't do them both at the same time easily.

- If you have a layer with two nodes then you could make a story for each path because there are only two paths. With three nodes there are 6 paths - still possible.

- The growth is factorial. If you have a layer with13 nodes… not possible.

# How then?

- So to create narrative and still have non-linear branching you must **isolate the sub-plots** into their own dependency tree.
- Follow the dependencies backwards to ensure your plot points are all within the inverted dependency tree.
- This is probably why Noah Falstein and Ron Gilbert advise to start at the ending and work backwards.

# Example

- The graph on the right shows that in this game the three major plots are almost completely isolated.
- Even within these sub-plots there is more isolation into sub-sub-plots.
- If you read the graph backwards you can make sense of the story. Reading it layer at a time, no.

# Narrative Dependency Diagram

- Untested theory - create a dependency diagram for narrative.
- Nodes would be anything which reveals a plot point
- Edges would be the plot points, narrative information
- The tree would represent multiple stories at once and could ensure that the narrative is consistent if the dependencies are satisfied.
- This might be able to be incorporated into the puzzle dependency graph also.

# In short

- Non-linear branching expands the graph horizontally.
- Narrative happens vertically.
- If you want puzzles to integrate with narrative you need to isolate the non-linear branching into manageable chunks
- This is exactly what DOTT does. See graph #3.

# Tools

- **Why use tools?**
    - Graphs look better. Also they have automated layout algorithms and analysis possibilities.
- General graphing tools:
- Visio
- Omnigraffle

# Tools 2

- Graphviz - simple common file format
- Gephi - designed for larger graphs. Includes spreadsheet like functionality for extra node properties.
- Tulip - Sugiyama layout. Python scripting.
- Yed - The best for this purpose. Good hierarchical layouts with lots of options. Swimlanes. Very easy to use. Was used to create these graphs.

# CAM

- Cambridge Advanced Modeler
- Can generate DSM
- **Can generate GANTT**
- Can assign time range to each node and use statistical modeling to play with difficulty.

# Summary

- Make a puzzle dependency graph when designing your game
- Don’t fear non-linear branching
- Isolate narrative subplots
- Work backwards
- Use tools
- Thanks!