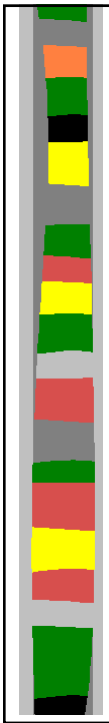
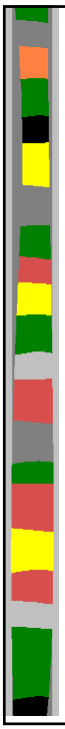


The Game Development Process

Game Architecture



Outline

- Tokens
- Initial Architecture
- Development
- Nearing Release
- Postmortem

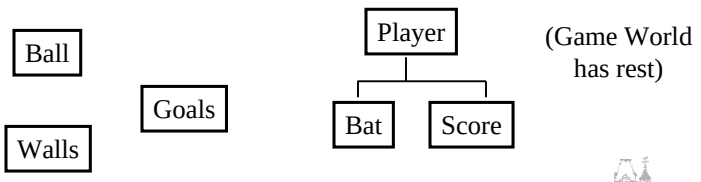
Game Decomposition

- Consider: *Pong, Frogger, Pac-Man, Missle Command, Zelda, Virtua Fighter, Warcraft, Doom …*
- What do they have in common?
  - All have a player (else a movie or screen saver)
  - All have discrete elements that can be directly or indirectly manipulated
  - Call these *tokens*
- Note, tokens == objects. Use "tokens", tho, since not always 1-to-1 mapping to software objects

Based on Chapter 17 of *Game Architecture and Design*, by Rollings and Morris

Tokenizing Pong

- Bats, Score, Ball, Walls
  - Player moves Bat, changes score so sub-tokens
- Goals, too. Defined by area.
- All games can be tokenized (*Pac-Man* and *Balls!* in Rollings and Morris book)



Based on Chapter 17 of *Game Architecture and Design*, by Rollings and Morris

Interactions of Tokens (1 of 2)

- Collisions are common
  - Token gets message telling collision occurred
- More interactions than collisions. Try token-token matrix (lower triangle, p. 484)
  - If impossible, "X"
  - If symmetric, square
  - If asymmetric, triangles
- Events: Ball-Bat deflection, Wall-Bat stop, Wall-Ball deflection, Goal-Ball goal event, Goal→Score goal event. Note, Score→Goal is "X"
- Allows visual check for interactions.
  - See errors, missing interactions
  - Maybe unexpected chain reactions (could enhance game, could be unplayable)

Based on Chapter 17 of *Game Architecture and Design*, by Rollings and Morris

Interactions of Tokens (2 of 2)

- Game World is token. Included in matrix. Needs to be informed of some events. Act as intermediary.
  - Also, objects don't need to know all they may encounter. Makes it easier to update.
- Ex: Ball hits goal → goal generates goal event to Game world → game world generates score event sends to score → score increments total points
  - Could later add team score or high score object and goal would not need to know

Based on Chapter 17 of *Game Architecture and Design*, by Rollings and Morris

Limitation of Token Matrix

- Can get complicated. Consider Pac-Man (Figure 17.15 in Morris and Rollings)
- Tokens have one or more states
  - Ghosts *hunting*, *hunted*, *eaten*
- Some interactions more complicated
  - Pac-Man eats power pill → power pill event
  - Power pill event → ghost goes to hunted, timer reset
  - Hunted ghost eaten → eaten event to home base, calculates how many → score (200, 400…) → score to ghost to display

State Machine for Ghost Token

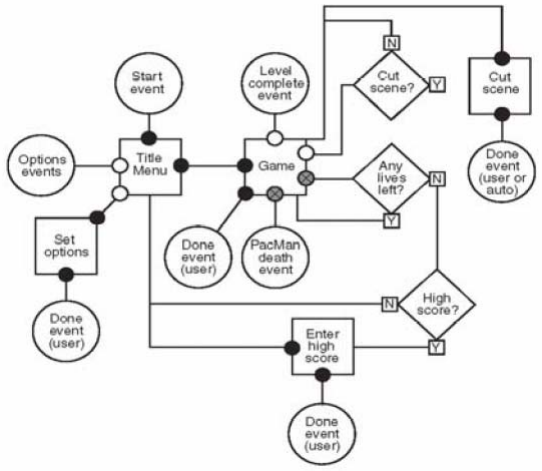
- Token Matrix gives you big picture
- Finite State Machine (FSM) gives you focus on specific area (Example of ghost in Figure 17.16)
- Single token and how rest of world interacts. Ex: ghost
  - Hunter – (pill) → Hunted
  - Hunted – (timer) → Hunter
  - Hunted – (pill) → Hunted (reset timer)
  - Hunted –(pacman) → Eaten
  - Eaten –(resurrect) → Hunter
- Eaten would trigger score event. That would appear in FSM of score token

State Machine for Pac Man

- Hunted – (power pill) → Hunter
- Hunter – (timer) → Hunted
- Hunter – (power pill) → Hunter (reset)
- Hunted – (ghost collision) → Dying
- Dying – (if lives > 0) → Reset Level event else Game Over event
- This is an "open" FSM, meaning can be a dead-end

Based on Chapter 17 of *Game Architecture and Design*, by Rollings and Morris

State Machine for Game World



- Shows how FSM translates to non game aspects
- Pretty generic for all games
- Would be others, such as score, etc.
- FSMs hierarchical, break down finer
- Once mastered, allow visualization of complex game

(Errata .. 17.18 is same as Pacman FSM)

Based on Chapter 17 of *Game Architecture and Design*, by Rollings and Morris

Initial Architecture Design

- Rollings and Morris, Chapter 20

Development

- Rollings and Morris, Chapter 21

Run Up to Release

- Rollings and Morris, Chapter 22

Postmortem

- Rollings and Morris, Chapter 23