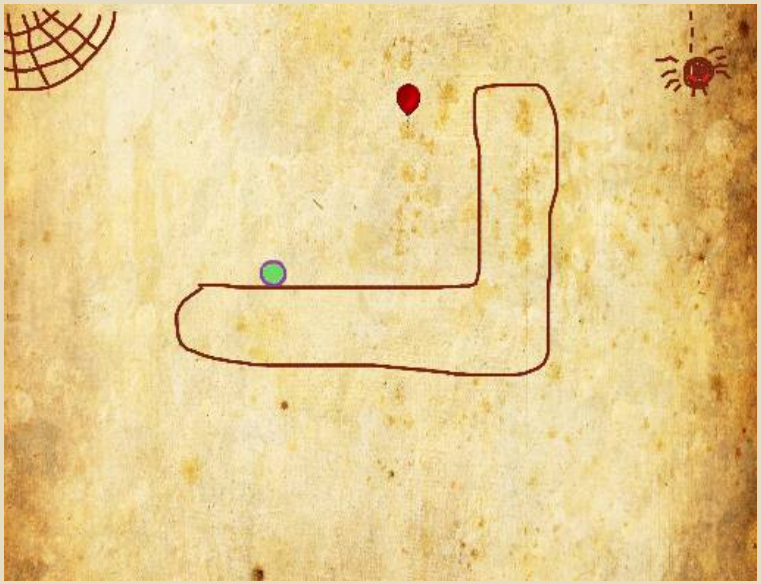
A Four-Process Implementation of Game-Based Scoring

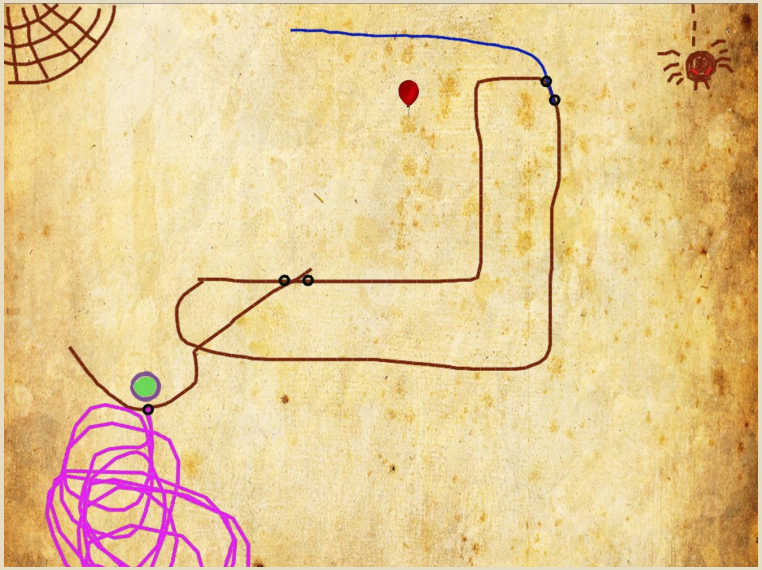
Russell G. Almond

Educational Psychology and Learning Systems
College of Education
Florida State University

Game Based Assessment Workshop, 2018

Physics Playground





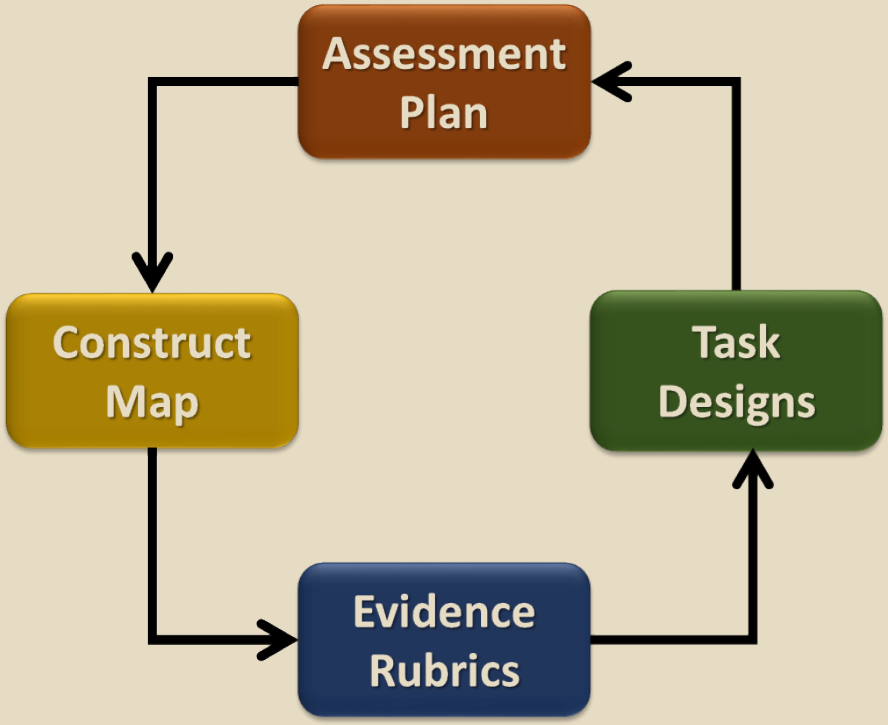
Example Level and Solution from *Physics Playground* Version 1. *Player Goal: Get the ball to the balloon by drawing ramps, levers, springboards and pendulums.*

Project Goals:

- Inference about conceptual physics
- Adaptive Sequencing of Levels
- Adaptive Provision of Learning Supports

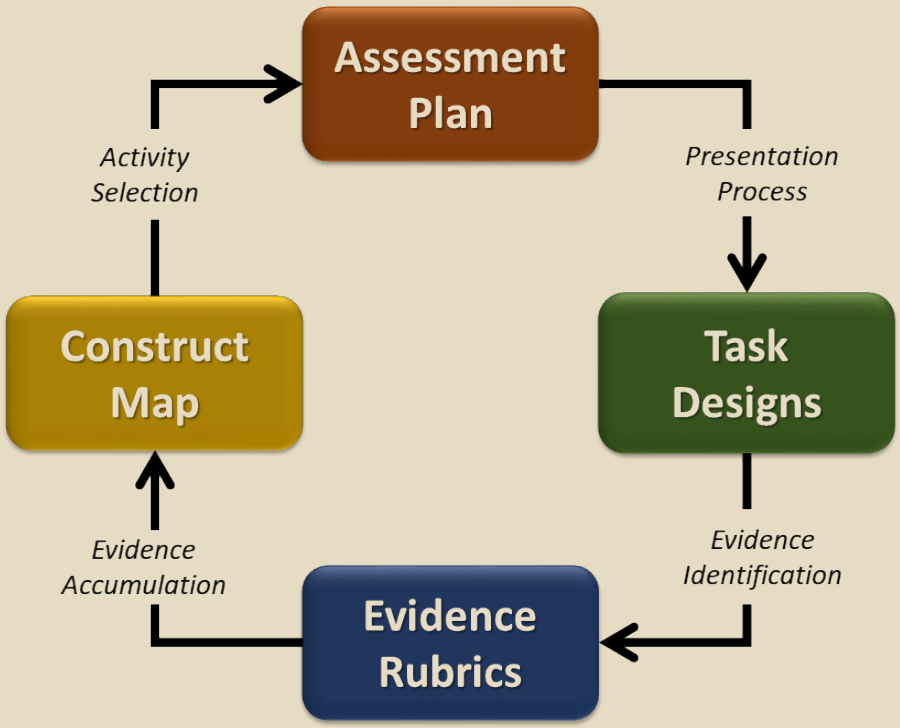
Need to infer state of physics understanding from game logs.

Four Elements of Assessment Design



1. Define a *Construct map* for the skills to be assessed (Proficiency/Competency Model)
2. Describe *evidence* that a student has the constructs (Evidence Model)
3. Create designs for *tasks* where the evidence can be observed (Task Model)
4. Create a (Assessment/Lesson) *plan* for those activities. (Assembly Model)

Four Processes of Assessment Delivery



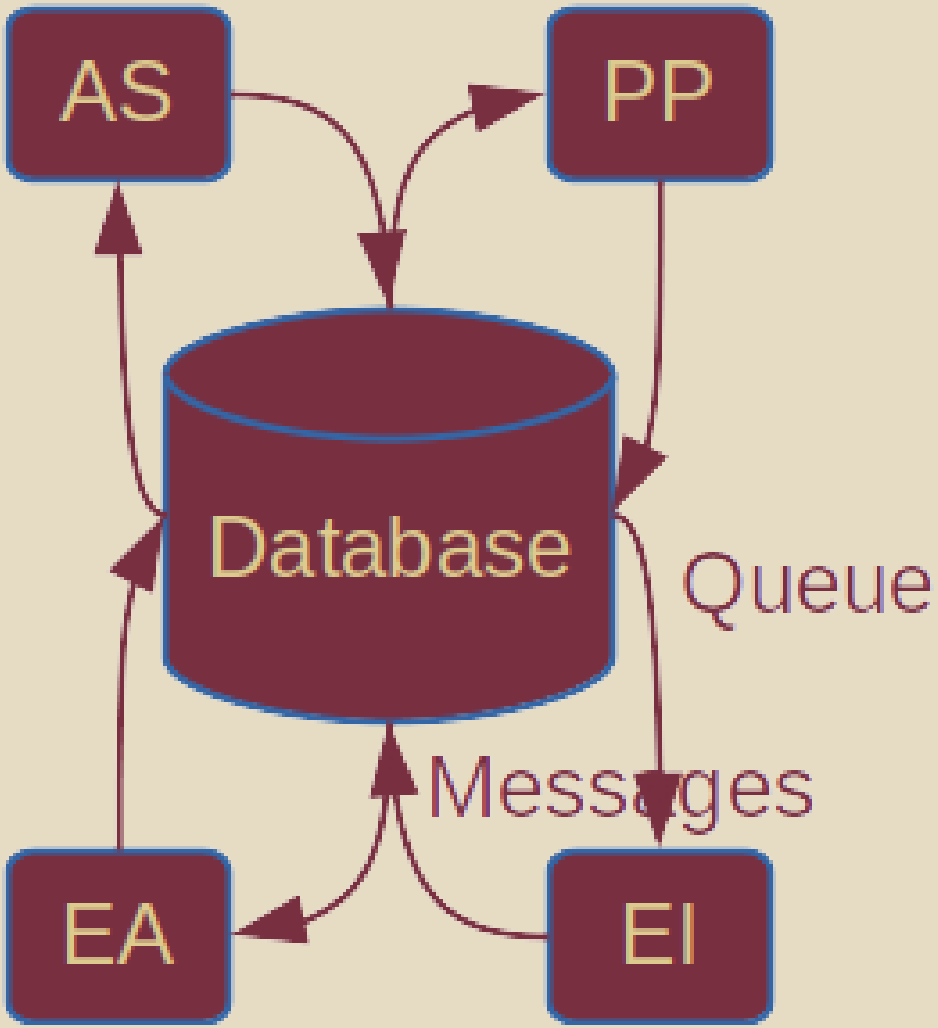
1. **Presentation Process (PP)** — Present the task and log events.
2. **Evidence Identification (EI)** — Extract key features (*observables*) from the stream of logged events.
3. **Accumulate Evidence (EA)** — Enter observed outcomes into the measurement model and locate player the construct map.
4. **Activity Selection (AS)** Based on player's current location, select next activity.

Proc4 Messages

```
{
  app: "ecd://epls.coe.fsu.edu/PP",
  uid: "Student 1",
  context: "SpiderWeb",
  sender: "Evidence Identification",
  message: "Task Observables",
  timestamp: "2018-10-22 18:30:43
      EDT",
  data:{
    trophy: "gold",
    solved: true,
    objects: 10,
    agents: ["ramp","ramp","
        springboard"],
    solutionTime: {time:62.25, units
        :"secs"}
  }
}
```

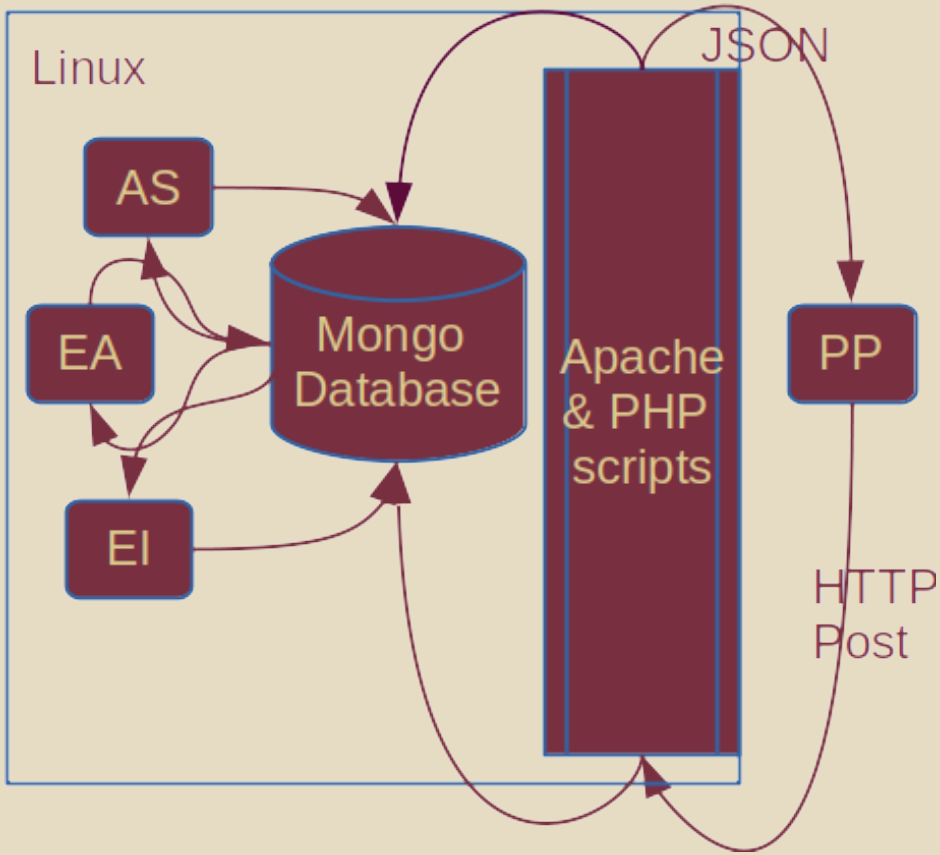
- Generic model of messages passed between processes (simplified xAPI)
- Application (app) header defines vocabulary used in other fields.
- Context equals task in this example.
- Data field can hold any number/kind of object.

Using a Database as the central message queue



- Document-oriented database (Mongo ®)
  - Fixed header for indexes.
  - Message-dependent variable data field.
- Database is also a Queue.
  - Add processed field.
  - Can sort by header fields
- 4 processes communicate by sharing database.
  - Each has input queue.
  - Post messages in other processes' queues.

Using Web Server for Communication to Clients



- What PP gets from server:
  - EI Game Status information.
  - EA Current Scores
  - AS Next game level
- Lightweight (PHP) process (Dongle) provides latest info from database.

Dongle Never Blocks. Always responds with most recent entry in database.

Game Engine: Unity and Learning Locker

- PP ran on different server (Unity Server) from EI, EA and AS (Scoring Server)
- Unity sends game bundle as package to browser which then runs it.
- Game client sends logging messages back to Learning Locker ®
  - Logs events in xAPI format.
  - Saves events in Mongo database.
- Game client communicates with scoring server through special URLs.
  - POST message with fields with Proc4 labels.
  - Return message as Proc4 JSON

Proc 4 Events versus xAPI

Event:

```
{
  app:"https://epls.coe.fsu.
      edu/PPTest",
  uid: "Test0",
  verb: "Manipulate",
  object: "Slider",
  context: "Air Level 1",
  timestamp:"2018-09-25 12:12:
      28 EDT",
  data: {
    objectType: "
        AirResistanceSlider",
    oldValue: 0,
    newValue: 5,
    method: "input"
  }
}
```

- Extension of Proc4 message format
  - Borrow *verb* and *object* fields from xAPI.
- Only *app* (application ID) is GUID.
- Only one extension container (*data*).
- Context is now ID for game level.
- Removes meta-data.

Learning Locker to Proc 4 Loop

- Server Process (on scoring server) takes events from records store into processing queue.
  1. Fetch all events from LL database since *timestamp*.
  2. Recode xAPI events as simpler Proc4 events.
  3. Update *timestamp* to last timestamp in message set.
  4. Filter out unused events.
  5. Load events into EI process queue.
  6. Small wait.
  7. Loop (until cows come home).

Rule Based Evidence Identification

- EI process is written as a rule-based system.
- Custom EIEvent language designed for processing events.
- State of each player is tracked:
  - Timers
  - Flags (variables which are not reported)
  - Observables (variables which are reported)
- Rules fire based on *verb*, *object*, *context* and other conditions (from state flags and event data).
- Trigger rules report observables to other processes.
  - Proc 4 messages are saved in database.
- Context rules determine when the "task" (game level) has changed.

Bayes-net Updating

- Scoring Model consists of:
  - Core Proficiency (Competency) Model
  - Evidence Model Fragments for each "task" (game level).
- When player first logs in Proficiency Model is copied to make player-specific Student Model.
- When observables arrive for level:
  1. Find student model for player
  2. Find evidence model for context (task)
  3. Adjoin student and evidence model fragment
  4. Instantiate observed variables and propagate evidence.
  5. Discard evidence model.
  6. Record monitored statistics
     - Marginal distribution of skill variable.
     - Expected level (score) on skill variable.

Activity Selection: Actual Implementation

- Game levels are split into topics
  - Each topic corresponds to skill variable in proficiency model.
  - Topics are sequenced by experts.
  - High, Medium, and Low groups within topic.
- Monitor score for skill variable.
  - If probability is above threshold: graduate to next topic.
  - If probability is below graduation threshold:
    - Give Low, Medium, or High difficulty task depending on skill variable score.
    - If out of tasks for group, give tasks from other groups.
  - If probability is below support threshold go into support mode:
    - Level launches with learning support.
- If run out of levels in a topic, move onto the next one.
- If player graduates from all topics, go to endgame.
  - Randomly present unplayed levels.

Experience from the Spring 2019 Field Test

- Approximately 250 students.
- 5 class periods of game play.
- Approximately 6,000,000 messages.
- Processing speed for EI was an issue:
  - Filtering!
  - Needs multi-threaded implementation.
- Processing speed for EA was an issue.
- Need better mechanism for coordinating among teams
  - Vocabulary/Naming for Levels
  - Vocabulary for Verbs and Objects in events
  - Vocabulary for Observables
- Rule-based EI takes lots of effort to code.

Software Frameworks

- Resources:
  - Game demo and level editor: `https://pluto.coe.fsu.edu/ppteam/pp-links`
  - Peanut and RNetica `https://pluto.coe.fsu.edu/RNetica`
  - Proc4, EIEvent and EABN `https://pluto.coe.fsu.edu/Proc4`
- Thanks:
  - National Science Foundation grant DIP 037988, Val Shute, PI.
  - Bill & Melinda Gates Foundation U.S. Programs Grant Number #0PP1035331, Val Shute, PI.
  - National Science Foundation Grant #1720533, Fengfeng Ke, PI.