The Ninth Annual Game Design Think Tank
Project Horseshoe 2014

Group Report: Turtles All The Way Down

Participants: *A.K.A. "Turtles All The Way Down"*
Mike Sellers, Indiana University
Isaac Knowles, Indiana University
Kyle Brink, Viggle
Ken Rolston, Warner Bros. Interactive
Ted Castronova, Indiana University
Nick Laing, Microsoft
Marc LeBlanc, Riot Games
Kenny Shea Dinkin, GSN
Zach Gage, Independent
Darren Malley, Vicarious Visions
Facilitator: Ron Meiners, Far Flung Socks

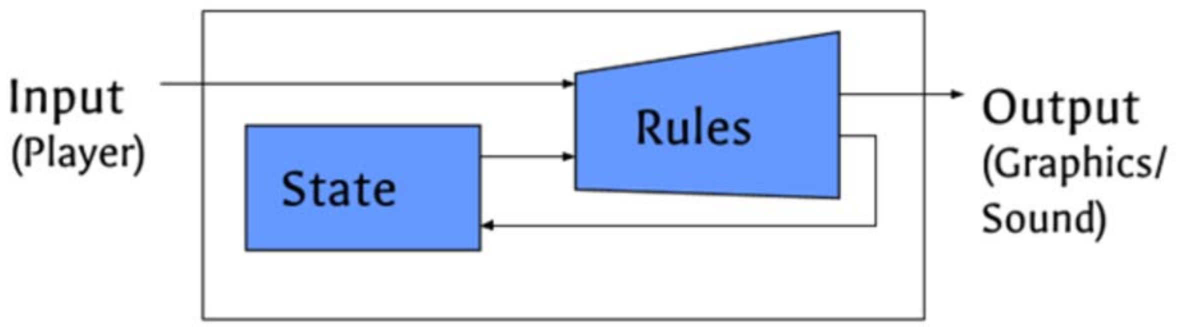
Problem Statement

Systems design is a vital craft in the games industry, but practitioners lack shared language and methodology. This has made it difficult to pass on knowledge to proteges, managers, and other team members, and it has stymied efforts to carry lessons learned from one project to the next. These issues in mind, Turtles All the Way Down had three goals:

1. To outline the basic concepts and abstractions of systems design
2. To agree upon a terminology, ensuring that it is practically useful, rather than formally precise.
3. To suggest ways to pass these lessons on to our colleagues and make it part of the designer's lexicon

We were able to accomplish 1 and 2. We've only scratched the surface of 3. We hope that this report, a compilation of useful concepts and abstractions for game systems design, provides a foundation for a community of systems designers to carry on the conversation from here.

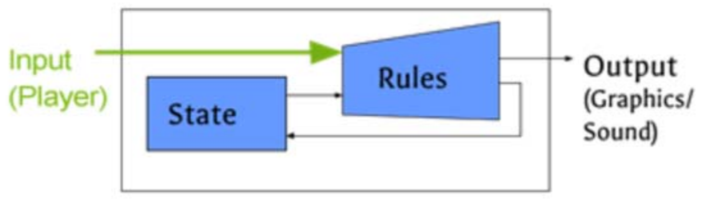
STATE MACHINE



A state machine describes a system with internal logic and state that changes over time. The parts of a

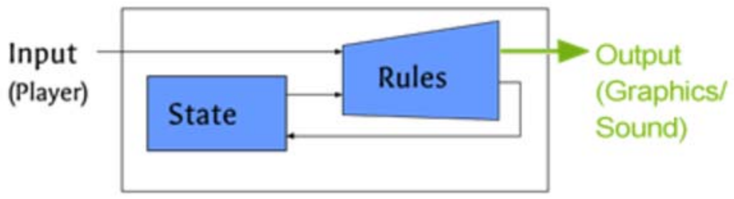
State Machine are as follows below:

INPUT



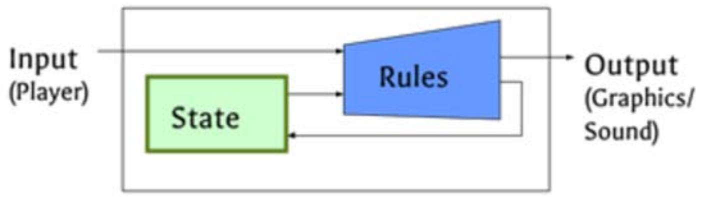
Signals received by the system from the outside - for instance, the player's control inputs.

OUTPUT



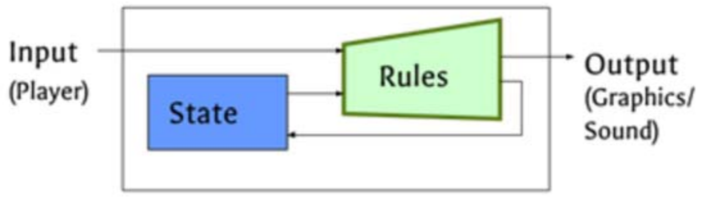
Signals sent by the system to the outside. These might drive graphics, sound, or any number of other game aspects.

STATE



The transient status of the system, which changes in relation to inputs, rules, etc. Another way to think of "state" is that it includes everything you need to save the game.

RULES



The internal logic of the system, which evaluates the current state and inputs to determine the outputs and the future state.

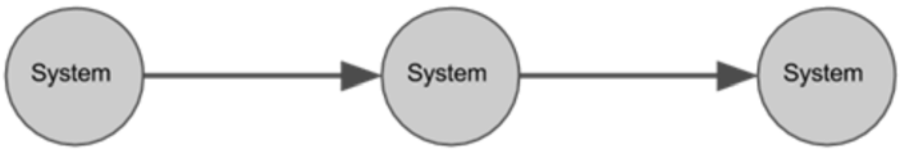
EXAMPLE STATE MACHINE: TIC-TAC-TOE

- **STATE:** Position of X's and O's
- **INPUT:** Player's next move
- **OUTPUT:** One of…
  - That move was illegal.
  - Here are the remaining legal moves.
  - You win!
- **RULES:** One mark per space, three-in-a-row wins.

COMPOUND SYSTEMS

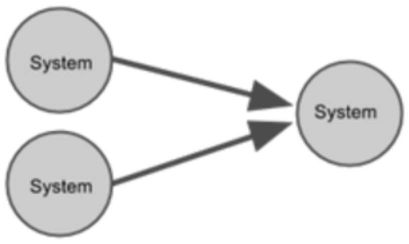
Compound systems are created by connecting the inputs and outputs of systems to one another. Examples include:

SYSTEMS IN SERIES



- Occur linearly, with the output of one system feeding into the input of the next
- Effects propagate down: a snowball effect is possible by building effects up within the system
- For example: carrying over of player units from one mission to the next in an RTS such as Homeworld

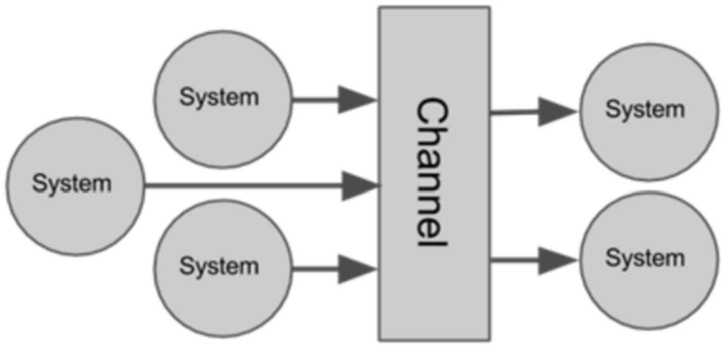
SYSTEMS IN PARALLEL



- Parallel systems run simultaneously with one another (as opposed to the chained inputs/outputs of serial systems)
- For example: quest rewards in MMORPGs such as WoW that can be acquired with multiple

resources - the player has a choice (more robust)

INTERACTION CHANNEL



- “Interaction channel” is the term used to describe the medium for anonymous communication between systems.

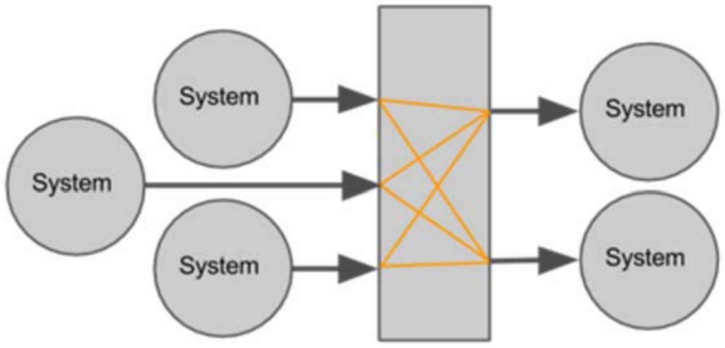
EXAMPLE INTERACTION CHANNEL: SOUND IN THE GAME “THIEF”

- AI responds to sounds.
- Sounds are created by multiple systems:
    - Movement
    - Combat
    - Lockpicking
    - Etc.
- This kind of interaction channel is referred to as an “event channel.”
- Amusing bugs or unintended behavior can occur as a result of unexpected communication over interaction channels: for instance, during the development of Thief a guard NPC heard the sound of a falling hammer and mistook it for the sound of combat, then fled the scene to summon reinforcements

RESOURCES AS CHANNELS

- It is possible for in-game resources to be considered channels: a resource creates anonymous communication between the *producer* systems and the *consumer* systems.

INTERACTION CHANNELS → EMERGENCE



- Interaction channels create the conditions for emergent complexity (described below), by creating many implicit communication pathways.

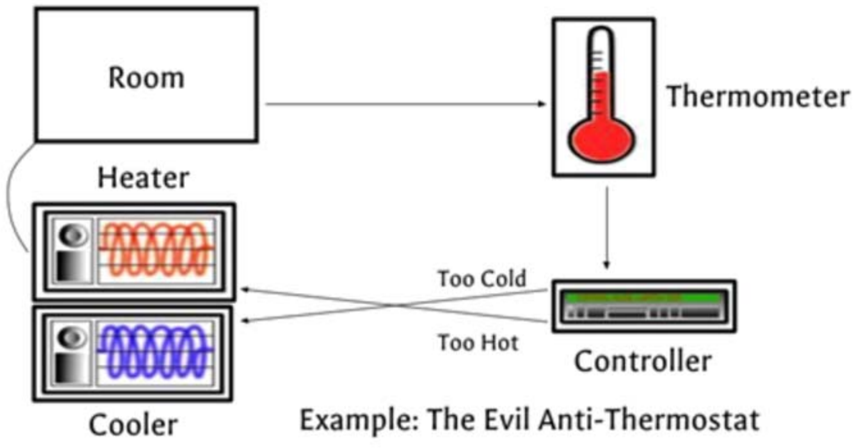
EMERGENT COMPLEXITY

- Also referred to simply as “emergence.” Emergent complexity constitutes properties that cannot be feasibly inferred from a system’s rules. These properties become apparent only during the game's runtime.
- For example: marketplaces in MMORPGs such as Everquest where players can sell goods – the offerings and interactions that result from such a system are not predefined, but rather occur dynamically as a result of emergence.

FEEDBACK SYSTEMS

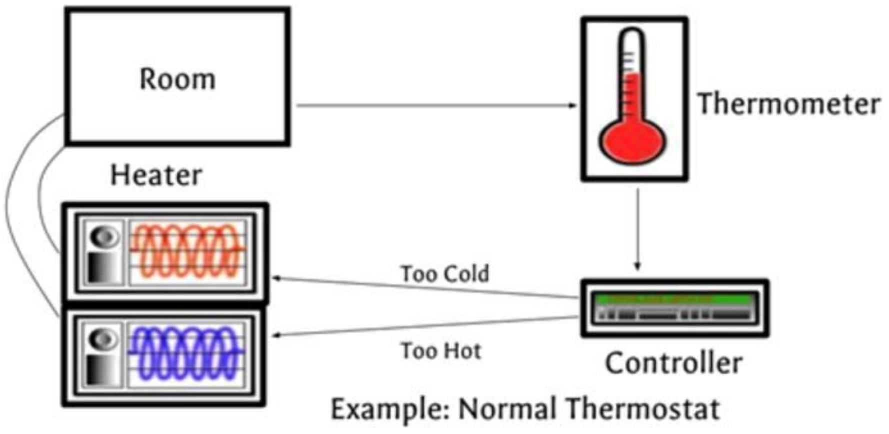
- A feedback system feeds its output back into its input. Generally these take the form of either a Positive Feedback system or a Negative Feedback system (outlined below).

POSITIVE FEEDBACK SYSTEM



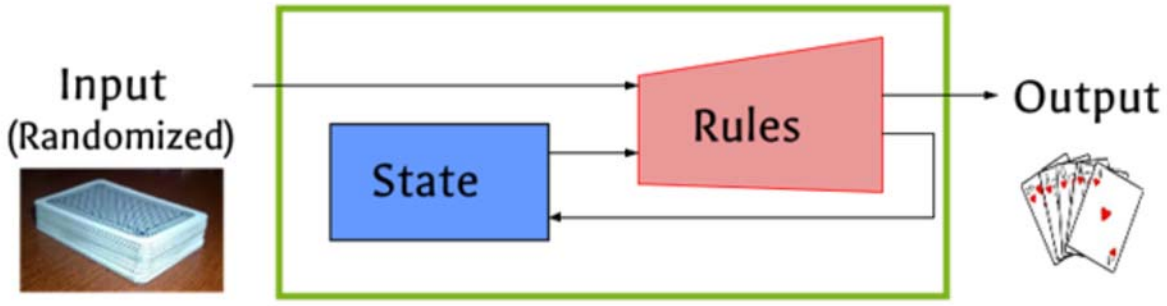
- A system that increases the difference between itself and a target value.
- For instance: money in Power Grid or Monopoly; the rich players tend to get richer and the poor players tend to get poorer

NEGATIVE FEEDBACK SYSTEM



- A system that reduces the difference between itself and a target value.
- For instance: the Blue Shells in Mario Kart – these create balance by causing players that are far ahead of the others to be brought back again

RANDOMIZED PROCESS



- A system with at least some random inputs. Generation of these inputs need not be opaque. As an example: turning over a card in Texas Hold 'em.

TUNING

- The act of bringing systems into alignment with game design goals by
  - examining and analyzing systems,
  - adjusting inputs and/or rules,
  - observing the outputs, and

    - repeating until game design goals are met.

CONCLUSION

We feel that the concepts outlined throughout this report are among the most crucial for designers to understand in order to begin communicating clearly with one another in the language of systems design. However, they are by no means the only useful ideas to be familiar with in such discussions. Depending on their needs, designers or educators using a framework such as this to describe game systems could add or subtract concepts to any degree of complexity. The next step we hope to achieve is to introduce our fellow developers who may not be familiar with them to these ideas, either in systems-related conversation, in formal presentation or perhaps both, in order to foster the kind of mutual understanding we would like to strive for.