

# *Gestural and Tangible Prototyping Tool*

CATGames Developer Interface

December 1, 2008

IAT 334  
D101, Team 1

Evan Miller  
Andrew Nip  
Nathan Waddington  
Shanna Walters

# *Table of Contents*

1.0 GENERAL INFORMATION	
1.1 Purpose	1
1.2 Project References	1
1.3 Terms and Abbreviations	2
2.0 SYSTEM CHARACTERISTICS	
2.1 System Architecture	4
2.2 Tasks	5
2.3 Design Specifications	5
2.4 Wireframes	8
3.0 TECHNICAL SPECIFICATION	
3.1 End Users	20
3.2 Development	20
4.0 SECURITY	
4.1 Control Points	21
4.2 Vulnerabilities	21
4.3 Safeguards	21

# ***1.0 General Information***

## ***1.1 Purpose***

The team is building a system that works with the Gestural and Tangible Prototyping system (part of the CATGames project), which interfaces between game designers and programmers and a physical prototyping system. This is particularly challenging as the Gestural and Tangible Prototyping tool is a collection of physical objects all of which have their own specific properties that need to be interfaced with in an intuitive manner. As the people we are targeting as an audience tend to be very time-constrained and may not be willing to spend the time to discover the functionality in the system it's particularly important to make sure the system can be understood and used rapidly.

Interface Prototype can be found at <http://www.sfu.ca/~eam1/Interface.html>

## ***1.2 Project References***

CATGames. (n.d.). CATGames: Creativity | Assistive | Tools. Retrieved September 18, 2008 from <http://www.catgames.ca>.

Gestural and Tangible Prototyping. Retrieved September 18, 2008 from [http://wiki.iat.sfu.ca/CATGames/index.php/Gesture\\_and\\_Tangible\\_Prototyping](http://wiki.iat.sfu.ca/CATGames/index.php/Gesture_and_Tangible_Prototyping).

Shaer, Orit et. al. (2004). The TAC Paradigm: specifying tangible user interfaces. *Pers Ubiquit Comput*, 8, 359-369

Wakkary, Ron and Marek Hatala. (2007). Situated play in a tangible interface and adaptive audio museum guide. *Pers Ubiquit Comput*, 11, 171-191.

## ***1.3 Terms and Abbreviations***

**CATGames:** Creativity Assistive Tools for Games Network; a research network consisting of The University of Western Ontario, Simon Fraser University, Credo Interactive, with Seneca College as the host organization. Its vision is to create innovative, leading edge technology tools for game production that support Canada's burgeoning games industry by accelerating the creative process, and expanding and enriching content environments and platforms. (CATGames Proposal, 2007)

**Input Devices:** A variety of items that can be used for rapid prototyping.

**Load:** This option allows users to load a previously saved file so that they are able to continue the editing process of creating rapid prototyping

**New:** This enables the user to create a new file to create new rules and games to be rapidly prototyped.

**Object(s):** Items that can be used by the rapid prototyping interface to measure the various properties (see below) to create new games.

**Operators:** these operators help users define whether rules and/or inputs are combined to reach a specific outcome.

- and; This enables the user to combine rules to meet a specific outcome.
- or; This enables the user to establish an output action if one of multiple rules are met.
- not; This enables the user to establish an output action if one of multiple rules is not met
- xor; "and or"; This enables the user to establish an output action if one or more rules are met.

**Outcome:** A means for users to establish messages/outputs to indicate to players whether specific rules or criteria are met for games.

**Output Devices:** A variety of items that can produce messages and outputs to communicate to users based on rules created. Output devices may include (but not exclusive to) audio, video or screen.

**Properties:**

- acceleration; Measures the change in speed of the object(s) over distance.
- bounces; Records the number of bounces of the object(s).
- face value; Determines the face of the object(s) that is facing up.

- orientation; Measures the X, Y, Z axis rotation of the object(s). If die/dice are being used, if 6 is up, then 1 is down, but which way are 2, 3, 4 & 5 facing?
- position; Measures the X, Y, Z axis coordinates of the object(s). Position in space.
- velocity; Measures the speed of the object(s).

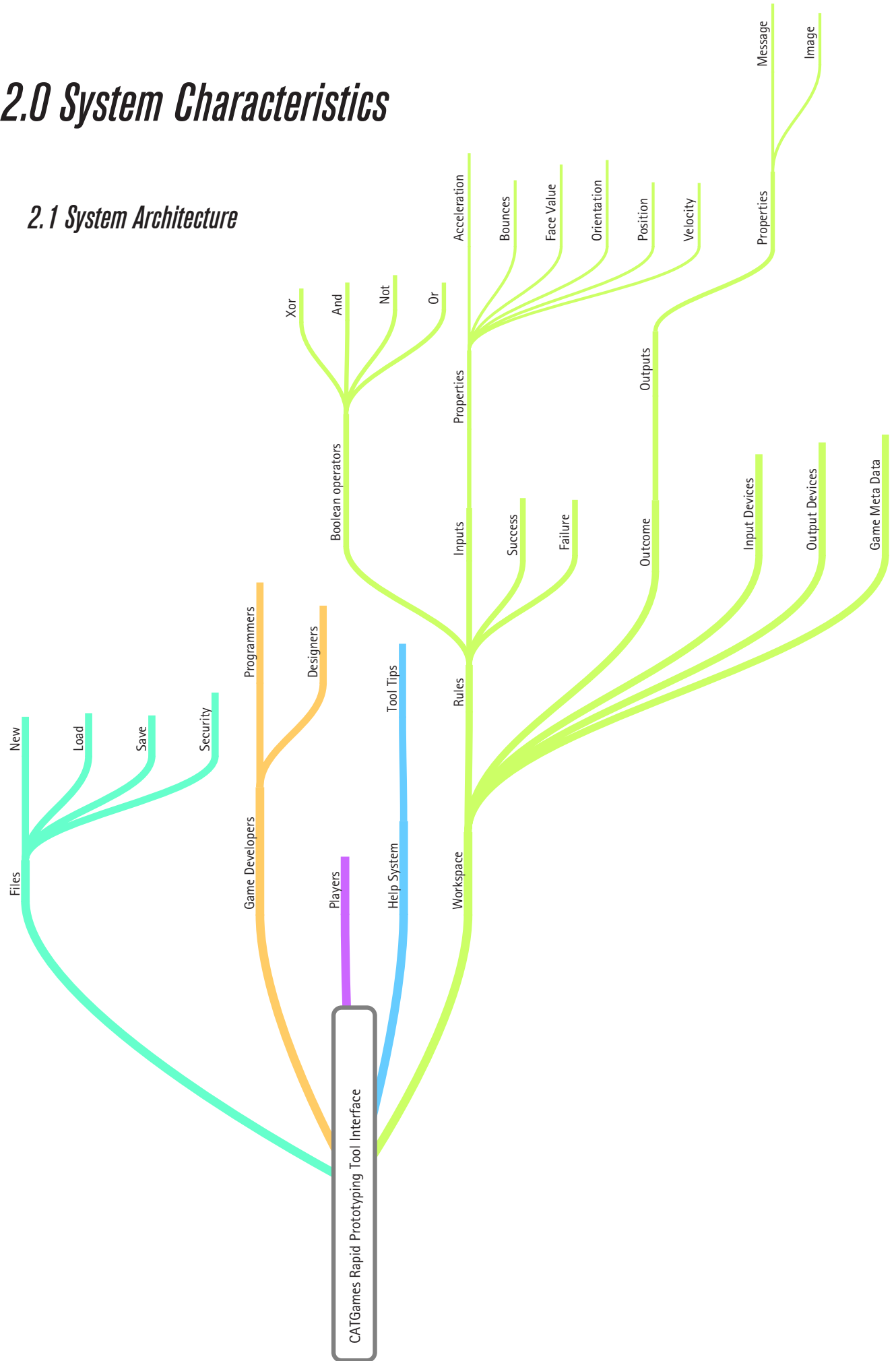
**Rule:**

- \* success
- \* failure

**Save:** Enables the user to save the current work file and reopen and edit this at a later time.

## 2.0 System Characteristics

### 2.1 System Architecture

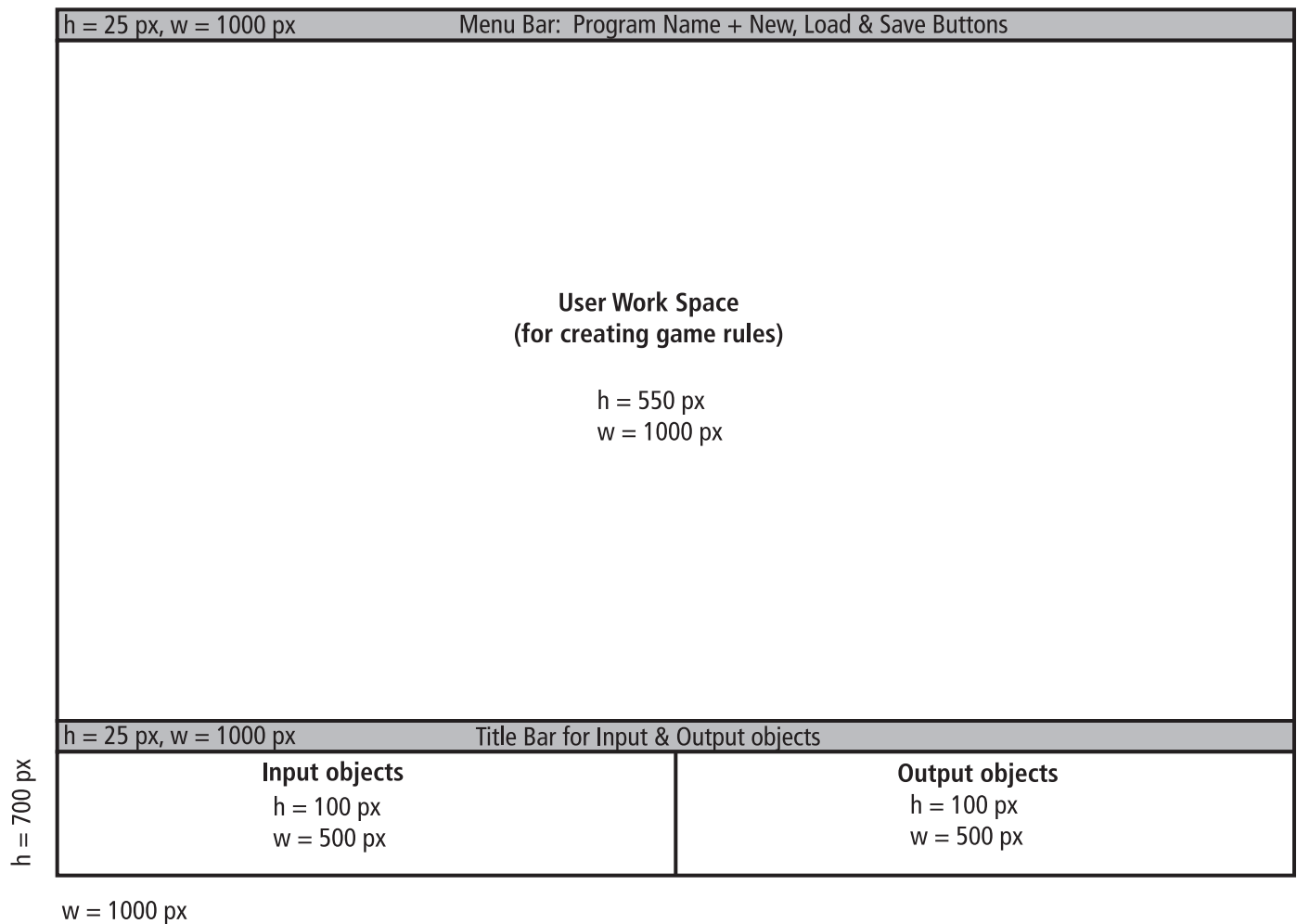


## 2.2 Tasks

1. Check for Snake Eyes and output text when the outcome occurs.
2. Create a Hot Potato Game.
3. Test for doubles and create different output text for when the outcome occurs and when it does not.

## 2.3 Design Specifications

Design Template:



## Style Guide

Font Usage:  
Calibri

Color Usage:



Outlines and Text



Workspace Color



Rule Box Color



Rule Box Highlight Color



Dice Icons



Screen Icons



Outcome box and Input and Output Menus Bars



Inputs, Properties, Conditions boxes and Grid Lines



Interactive Highlight Outline Cue



'AND' and 'Success' Connector Color



'OR' Connector Color

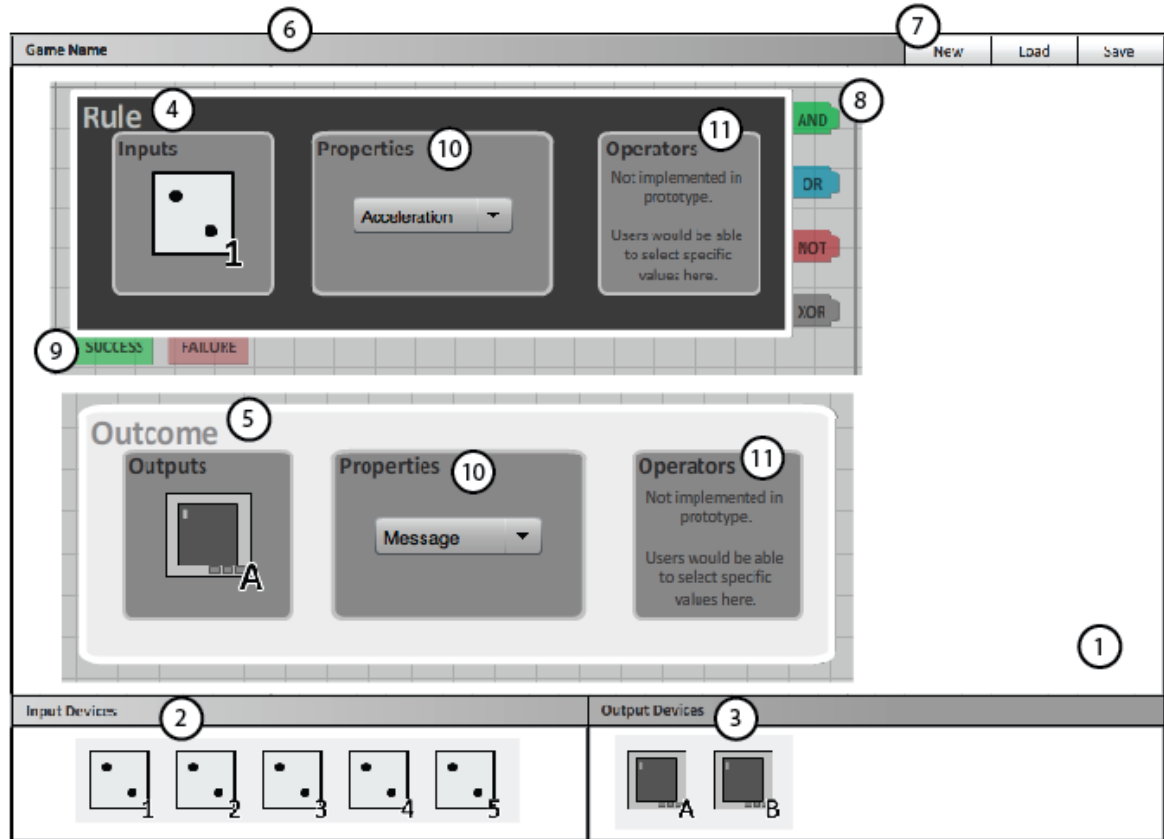


'NOT' and 'Failure' Connector Color



'XOR' (Exclusive Or) Connector Color

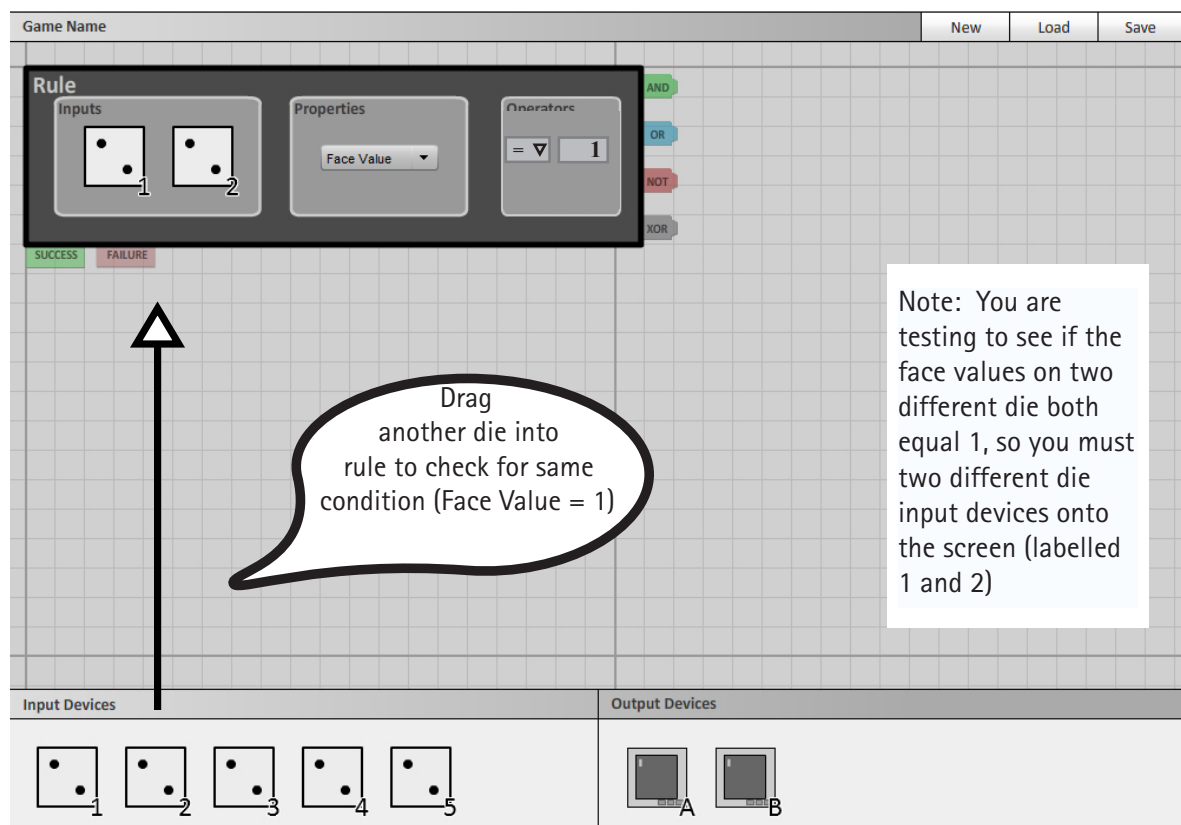
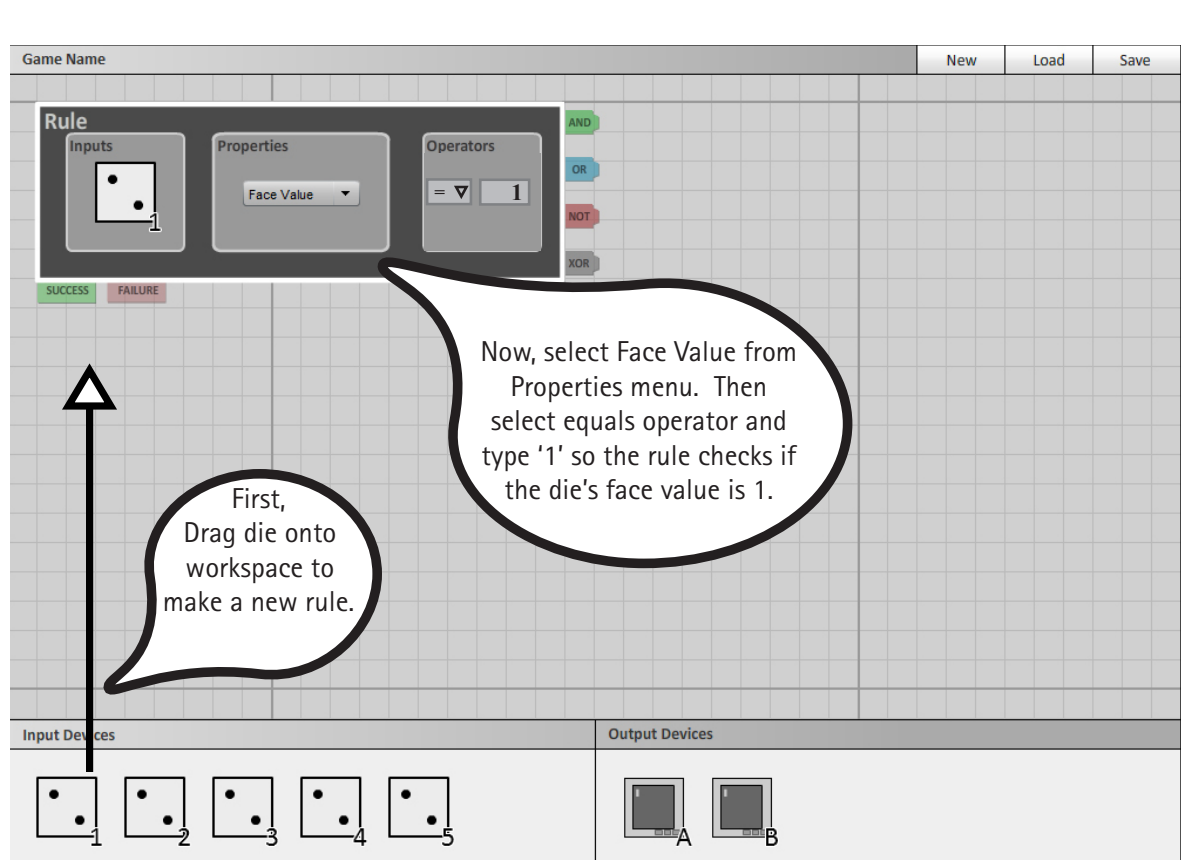
## Design Template:



1. Work area; shows any objects which have been placed. This is where all of the relationships between objects are shown.
2. Input devices; tangible inputs connected to the system (these need not only be dice, nearly any object with the appropriate sensors could be connected by the developers).
3. Output device; tangible outputs connected to the system (these need not only be screens, these could also be things such as speakers, motors, force feedback systems, etc.).
4. Rules; game based conditions which rely on Inputs, Properties and Operators (as well as other Rules in some cases) to establish the game play.
5. Outcomes; when a Rule's condition is met (or not met), then an outcome occurs, these rely on Outputs, Properties and Operators (as well as Rules) to determine how information is relayed to the player.
6. Meta-data; this is where developers can name games
7. File control; this is where developers can save, load, and create new games. Version control is available through this area.
8. Boolean operators (condition checks); boolean operators determine how rules interact with each other. This allows for complex games to be created and tested.
9. Rule output links (true/false, pos/neg); The Rule Output Links are what control how the Outcomes behave, if, during game play, a rule resolves true, then whichever Outcomes are connected to the Success link will be executed. If a rule resolves false, then whichever Outcomes that are connected to the Failure link will be executed.
10. Properties; attributes of objects (either inputs or outputs). Each object has it's own specific
11. Operators; users would be able to choose specific values for any property for any object, this allows for programmatical control of the system.

## 2.4 Wireframes

Check for Snake Eyes.

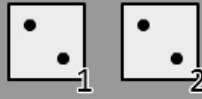


3

Game Name New Load

### Rule

Inputs



Properties

Face Value ▾


Operators

= ▾ 1

SUCCESS FAILURE

### Outcome

Outputs



Properties

Message ▾

Operators

Snake Eyes Have Occured!


Drag the output device Screen onto the page. Select the 'message' property and type the text output 'Snake Eyes have Occured'

Input Devices Output Devices

4


### Rule

Inputs



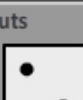
Success

Find the "Success" rule output link. From here connect the rule to the outcome.



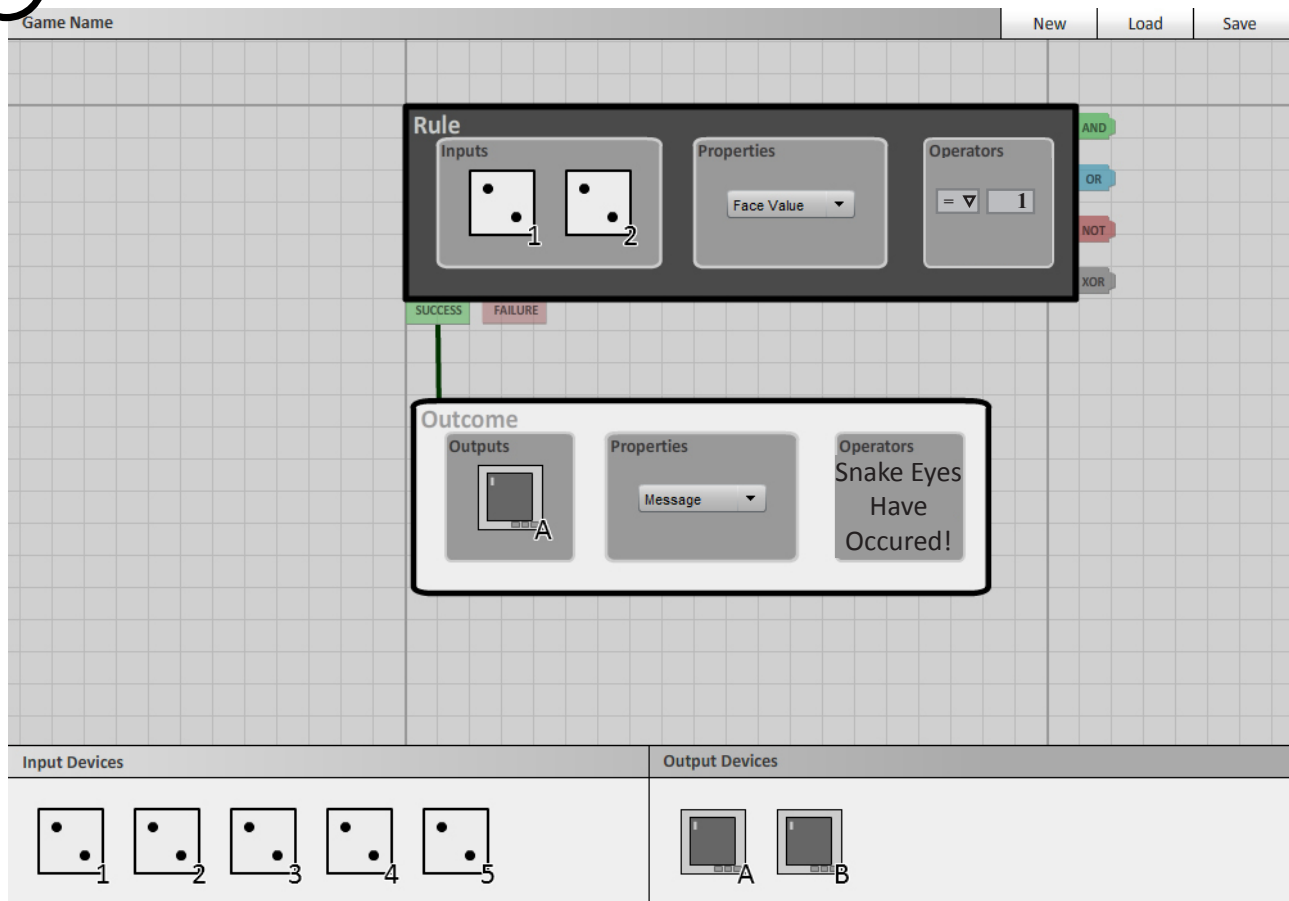
### Rule

Inputs



5

Finally the rule is done! Your workspace should resemble the following:



## Create a Hot Potato Game

Create new game to begin.

Game Name New

1

First Objective:

Set rule to check if a player is out of the game because they held the hot potato for too long.

This can be determined if the object is not moving (velocity = 0) but it is still off the ground (distance doesn't = 0).

Note: Imagine the die input device is a general object, such as a ball.

Input Devices: 1 2 3 4 5

Output Devices: A B

Game Name New Load Save

2

Rule

Inputs: 1

Properties: Velocity

Operators: = 0

SUCCESS FAILURE

AND OR NOT

Drag die onto workspace to make a new rule.

Select Velocity from Properties menu. Then set Rule to check if Velocity = 0 by selecting the '=' operator and typing 0.

Input Devices: 1 2 3 4 5

Output Devices: A B

3

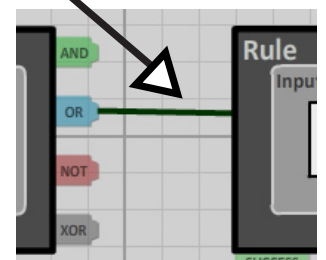
Drag the same die icon onto the workspace to create a second rule for it, testing this time for its position.

Select Position and set y to equal 0 (this will tell you if the item is on the ground)

However, you first want to test that the y-value (height) is NOT equal to 0 (indicating the object is not on the ground)...

4

Find the 'Not' Operator on the first rule's box and connect it to the second.



5

Game Name

New Load Save

Properties Face Value Operators = ▾ 0

AND OR NOT XOR

Rule Inputs 1 Properties Position Operators X = ▾ Y = ▾ 0

SUCCESS FAILURE

Outcome Outputs A Properties Message Operators Hot Potato! User is out of game.

Drag the output device Screen onto the page and create the output you want to appear if this rule occurs; create it so text appears saying 'Hot Potato! User is out of game.'

Input Devices

Output Devices

1 2 3 4 5

A B

6

Inputs 1

Success

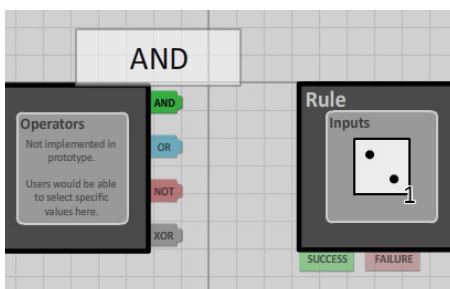
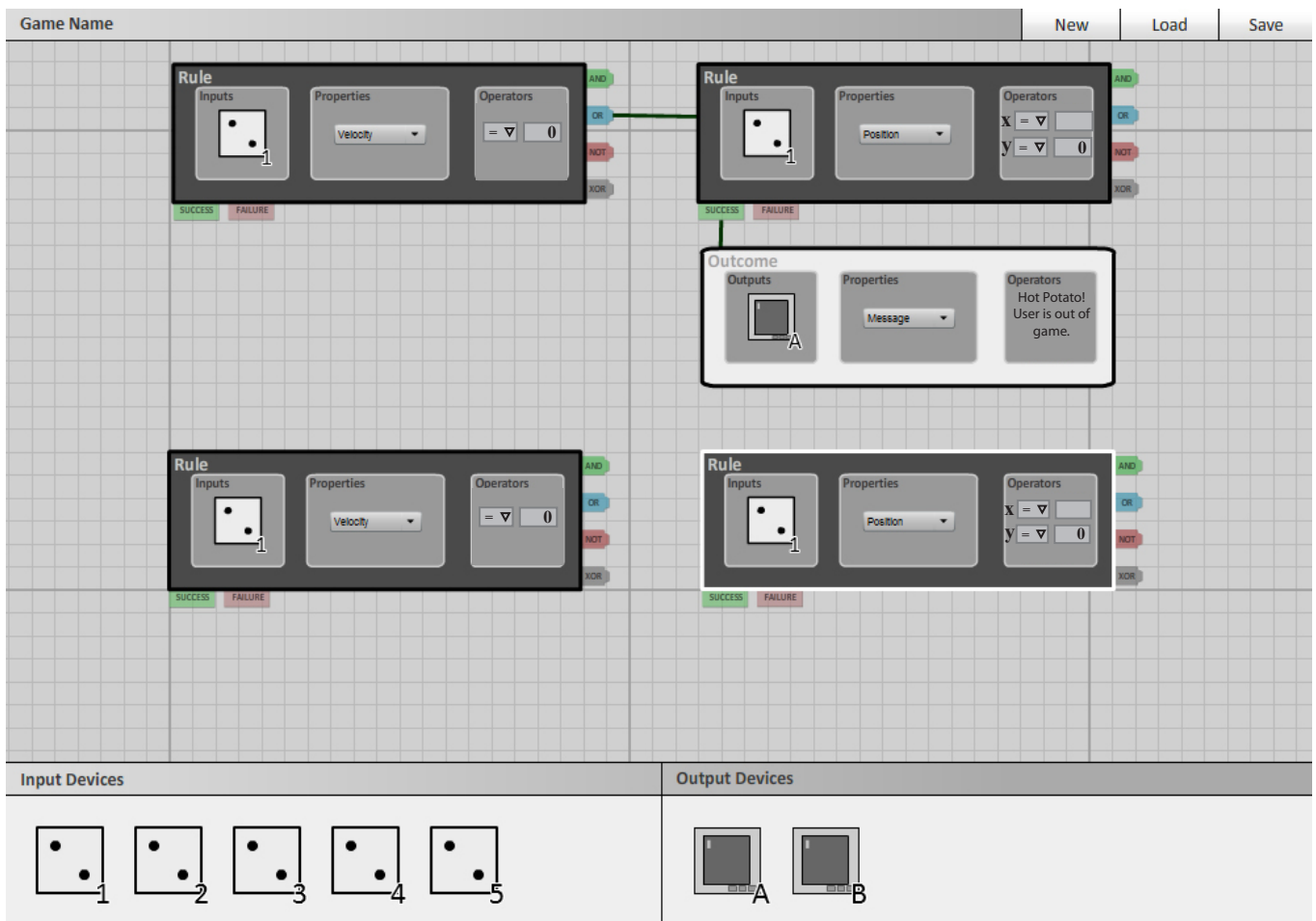
SUCCESS FAILURE

Outcome Outputs Properties

Find the "Success" rule output link. From here connect the rule to the outcome.

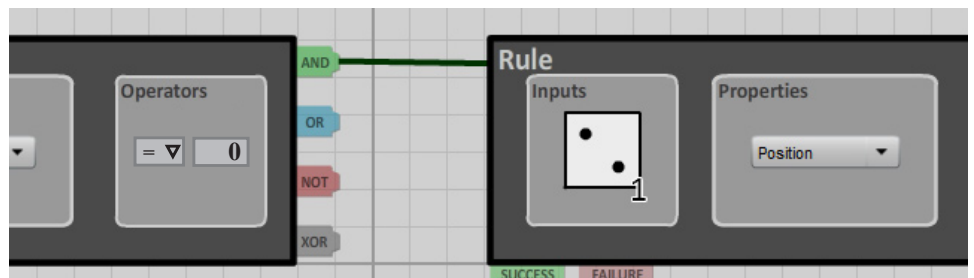
7

Repeat steps 2 through 3 on same work space in order to create rule to check if the game is over.



8

However, in step 4 connect with an 'AND' operator instead of the 'NOT' because you want to detect if both the  $V = 0$  and  $y$  position = 0 conditions are true. If both these conditions are true, then the 'hot potato' is on the ground and not moving, indicating the game is over.



9

Game Name New Load Save

**Properties** **Operators**

Velocity = ▾ 0

**AND** **OR** **NOT** **XOR**

**Rule**

**Inputs** **Properties** **Operators**

1 Position ▾ X = ▾ y = ▾ 0

**SUCCESS** **FAILURE**

**Outcome**

**Outputs** **Properties** **Operators**

A Message ▾ Game Over!

Drag the output device Screen onto the page. Choose to have the output as a message. Enter the text "Game Over!"

**Input Devices** **Output Devices**

1 2 3 4 5 A B

10

**NOT** **XOR** **Success** 1 **SUCCESS** **FAILURE**

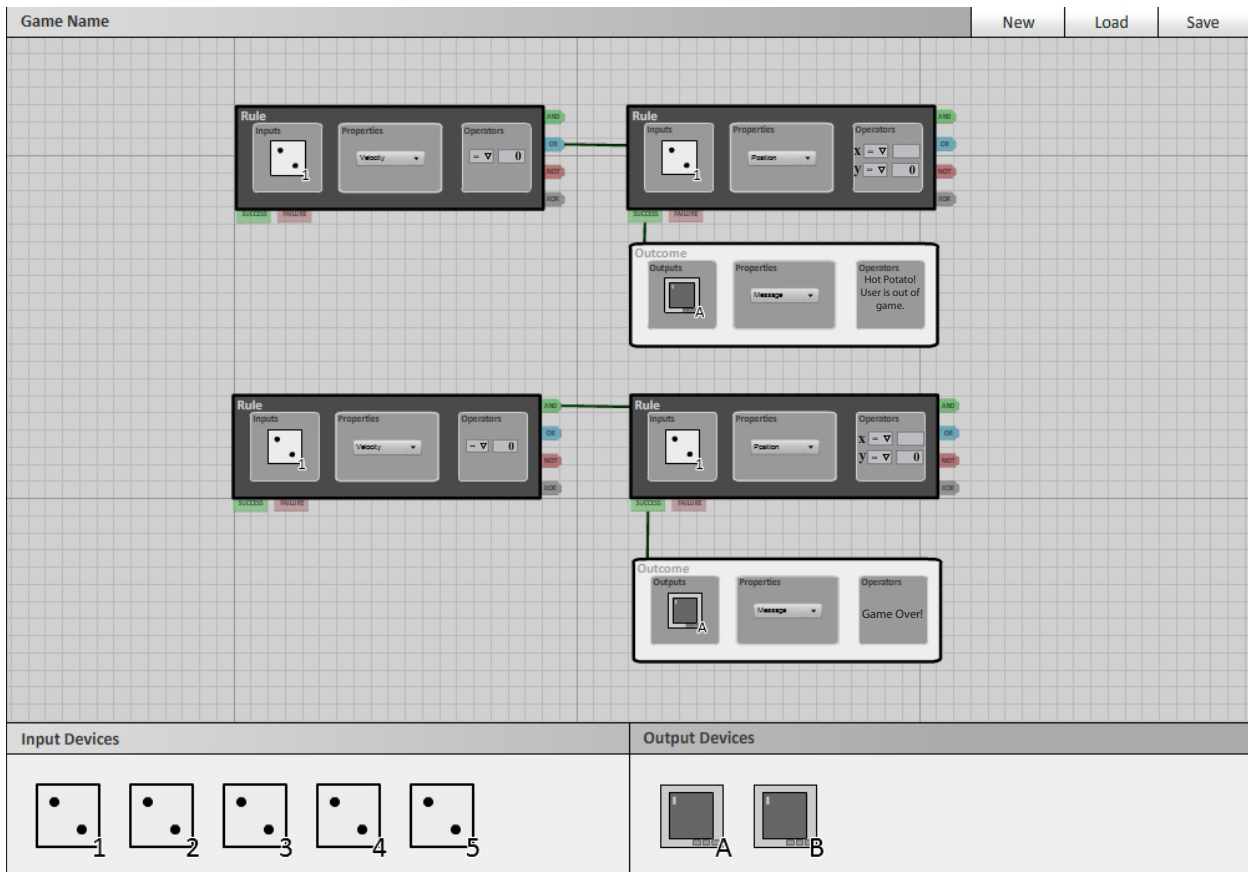
Find the "Success" rule output link. From here connect the rule to the outcome.

**Outcome**

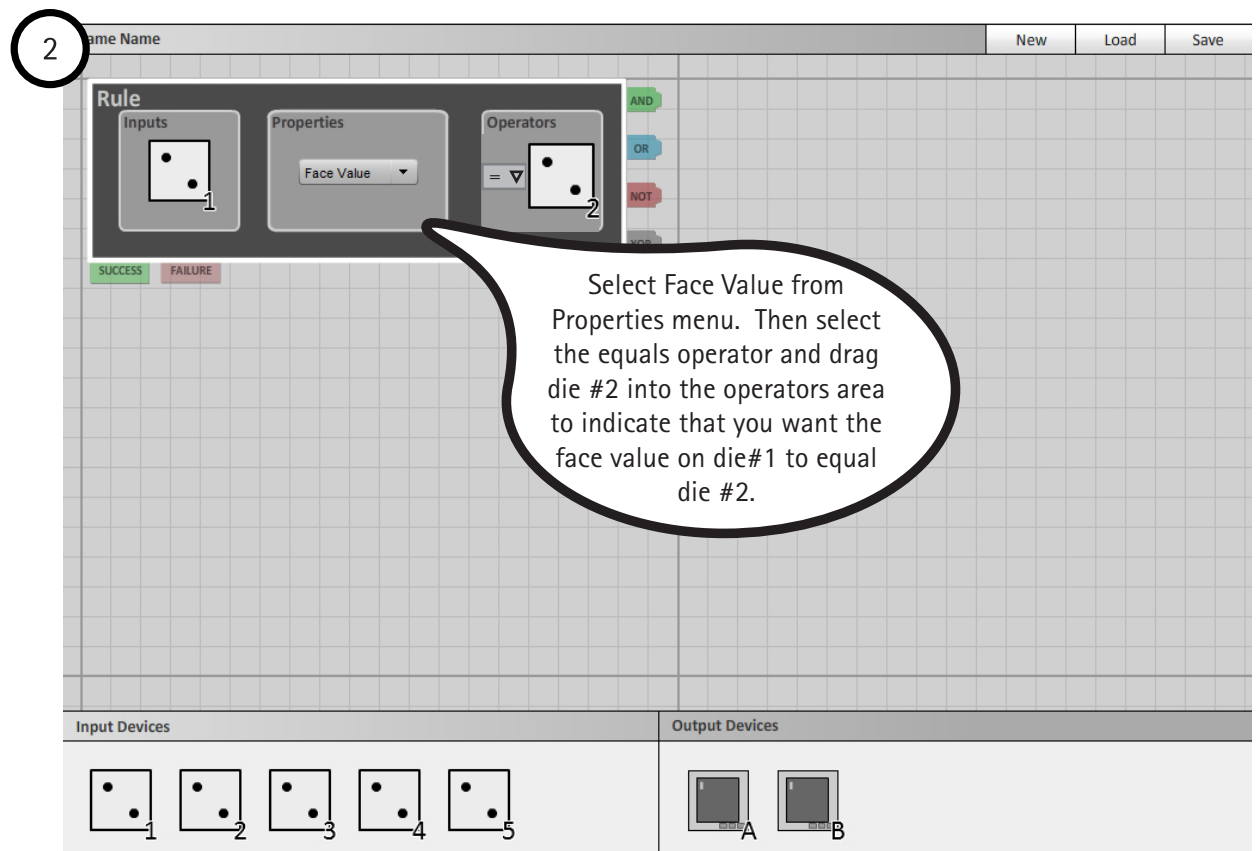
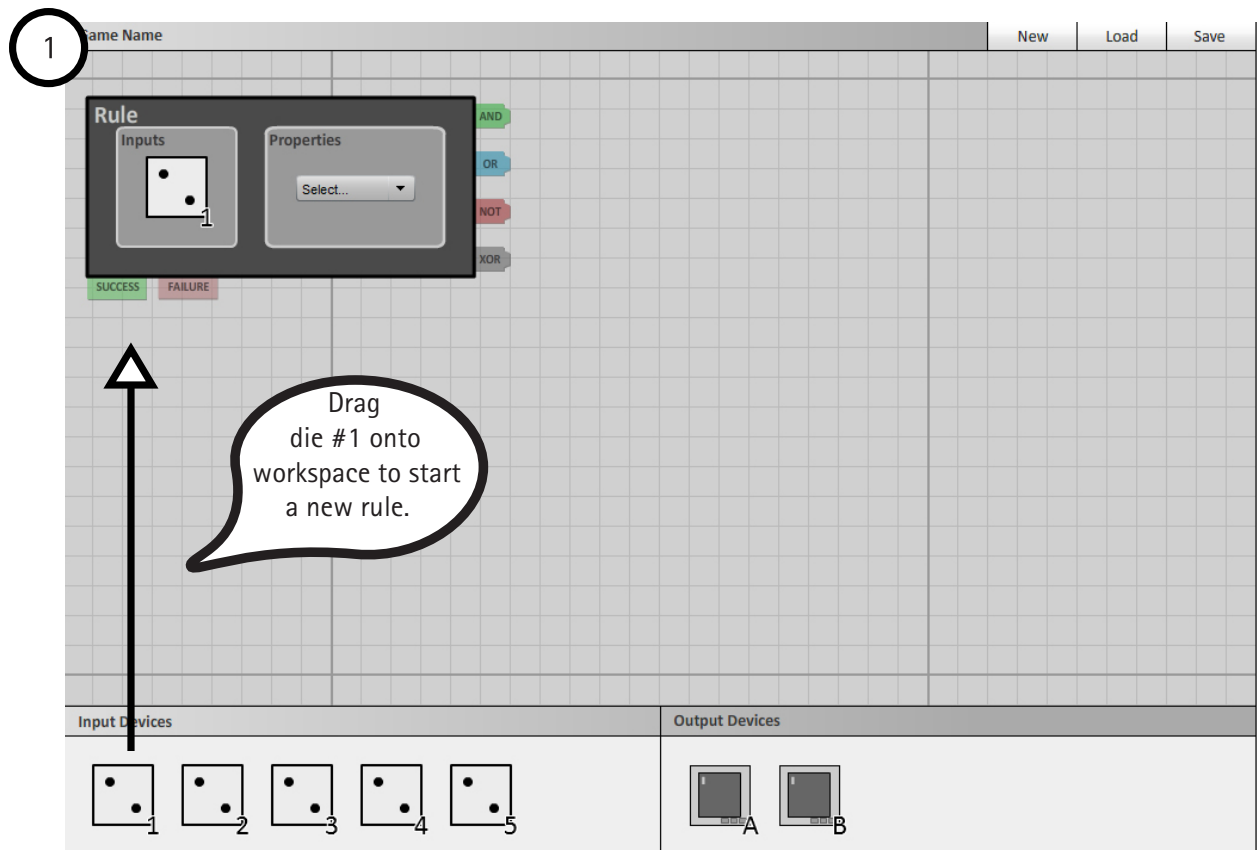
**Outputs** **Properties**

A Message

All Done! Your workspace for the game 'Hot Potato' should now look like the below:



## Check for Doubles and Create Positive and Negative Outcome Messages



3

Rule

Inputs: 1, 2

Properties: Face Value

Operators: =

Outcome 1: Outputs: A, Properties: Message, Operators: You rolled doubles!

Outcome 2: Outputs: A, Properties: Message, Operators: You did not roll doubles.

Output Devices: A, B

1. Drag two of the same output device screen icons onto the page so that you can create an output for both positive and negative occurrences of the rule. These will appear on the same screen.

2. For both outcome boxes select property type 'Message'. For the output box on the left make the message 'You have rolled doubles'. For the message on the right, make the message 'You did not roll doubles'.

Rule

Inputs: 1, 2

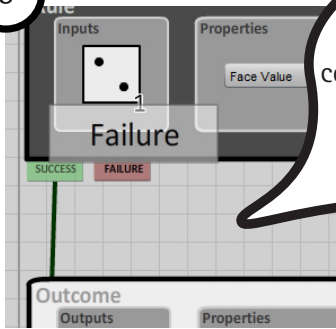
Success

Outcome 1: Outputs: A, Properties: Message, Operators: You rolled doubles!

Outcome 2: Outputs: A, Properties: Message, Operators: You did not roll doubles.

4. Find the "Success" rule output link. From here connect the rule to the first outcome.

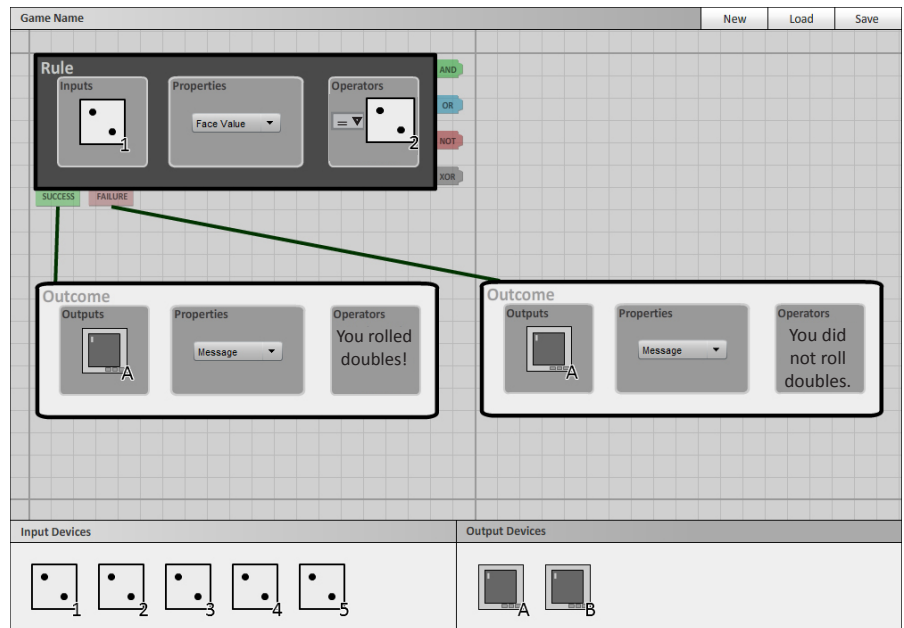
5



Find the "Failure" rule output link. From here connect the rule to the second outcome (with the feedback for negative occurrence of doubles).



Your final workspace should resemble the following:



## ***3.0 Technical Specifications***

### ***3.1 End Users***

The end user of our interface is intended to be game developers, who can be divided down into two particular categories – designers and programmers. Designers are those involved with the conceptual ideas such as overall concept, rules, character designs and so forth – the ideas that drive the development of the game. Programmers may be involved in the design stage of a game but their role is primarily for carrying out the concept into a working form.

### ***3.2 Development***

The current program that the CATGames group is using to develop programming is Max/MSP. The reason for this decision is that Max/MSP is "a graphical development environment for music and multimedia"(Wikipedia). Since the software was designed for interfacing with a variety of inputs and outputs, it is a logical choice for CATGames to use for communicating between the various input devices and output devices that it employs.

For our purposes for this course, we chose Flash to help prototype and develop our interface for a number of reasons. Team members' knowledge and familiarity with Flash provides quick and rapid prototyping of interfaces and iteration of designs. Most importantly, Flash can be connected with Max/MSP on the back-end of the development, enabling the interface to be connected to the program. This will enable any iteration of the interface to be connected to the system, tested and any issues regarding its integration and flow. Also, Flash is appropriate for our intended audience.

## *4.0 Security*

### *4.1 Control Points*

The main point of vulnerability is the file system. The system needs to be able to handle a developer saving a file, loading a file, and getting what they expect out of the process, this could break down if there are multiple users developing the same game. Another point to be aware of is version control, that is, if a developer makes a change, they should be able to go back to a previous iteration of the game, that way they can prototype with out worrying about whether or not new additions will break the game.

### *4.2 Vulnerability*

When programming games, changes can cause unintended effects. Changes to one section of the game rules could adversely affect other parts of the game. Data loss can also occur if the system is able to be used by multiple people at once (one programmer could overwrite a file that another programmer has changed -- losing the original programmer's changes).

### *4.3 Safeguards*

Building a game repository could alleviate some of the vulnerabilities. Set up a checkout system that forces users to check out individual files, and check in tested code when they are finished could help to prevent errors being introduced when changes are being made to a file. If the repository is on a remote system, this could also protect against developer system failure (the Tool would run on the developers machine, and the repository could run on an off site server). A file repository would also provide a versioning system, where changes that are not wanted could be rolled back, allowing the developers to move to previous versions which may be more appropriate than a more recent version. If the system is being used by multiple developers at once, the file repository with a checkout system would keep multiple developers from editing the same file at the same time, hopefully preventing data loss when both developers save their file (one overwriting the other).