### SandCat — Update 1

Welcome to the first update for my latest project, SandCat. I have not yet written a true introduction to the project, so you’ll just have to bear with me for now. But — here is the meat of the project. SandCat is the first game ruleset prototyping language. The language aims to expedite the game design process by allowing designers to focus entirely on the core mechanics and rules while ignoring the software requirements. If you want more information on what I mean by this, go read my previous article [Improving Game Design Process](https://medium.com/@ryanrothweiler/improving-game-design-process-9e392ebf73af).

This is the first update — obviously that implies this is the first of many. These updates are where I plan to discuss language design decisions, planned features, general progress, and really anything else that applies to the project. The updates will be quick, informal, and unpolished. I’m writing these for two reasons: 1. For the reader — to market the project, get the word out, and hopefully get feedback. 2. For myself — to document progress and organize my thoughts.

Considering these are quick informal updates, everything is first draft and will most likely be changed. Therefore: I strongly encourage the reader to ignore the minute specifics and focus on the long term goal. For example, don’t pay any attention to syntax. I’m not yet worried about syntax. I know it is very rough, sub-optimal, and wildly inconsistent. I aim to prove the core ideas before I refine the specifics.

### Groundwork

For this first update, I’ll simply introduce the beginnings of the project.

First off, all games needs some kind of state. You can specify state like such,

Chicken.

When played, you get this.

This is simply a truth statement. It’s difficult to conceptualize without more context. We can add some context by supplying a better name.

ChickenHealth.

But a chicken’s health implies a numerical value. We can specify that also.

ChickenHealth(10).

Great. We have some very basic initial state. Now how do we change that state? For now, that is done through events. They follow the pattern of

// event(EVENT\_NAME) - STATE\_CHANGING : CONDITIONS  
event(HurtChicken) - ChickenHealth(ChickenHealth - 1) : conditional(ChickenHealth > 0).

In english, this reads as — this event is called HurtChicken, it sets ChickenHealth to one minus the current ChickenHealth, and it is only possible if ChickenHealth is greater than 0.

I think that is a nice start. It’s very rudimentary, but it’s already capable of expressing simple games.

### Shield Push

Using just these two features, I was able to write a small game called Shield Push.

This game features a graphical representation of the rules. Don’t worry about that. It is very important to give designers tools to visually organize their game, but I haven’t tackled that challenge yet.

Back to the game. In this game the player can move the red square around, but never outside the bounds of the grid, and never into the black square. The blue rectangle is a shield which, once picked up, allows the player to push the black square only up or down. Below are the rules definition for the game, followed by a gif of the game being played.

// Initial state  
GridMaxX(10).  
GridMinX(0).  
GridMaxY(10).  
GridMinY(0).

PlayerX(2).  
PlayerY(3).

BlockX(3).  
BlockY(3).

ShieldPosX(5).  
ShieldPosY(7).

// Moving player. Can't move into the block.   
event(MoveUp) - PlayerY(PlayerY - 1) :   
conditional(PlayerY > GridMinY) & ! [ conditional(PlayerY - 1 = BlockY) & conditional(PlayerX = BlockX) ].

event(MoveDown) - PlayerY(PlayerY + 1) :   
conditional(PlayerY < GridMaxY) & ! [ conditional(PlayerY + 1 = BlockY) & conditional(PlayerX = BlockX) ].

event(MoveLeft) - PlayerX(PlayerX - 1) :   
conditional(PlayerX > GridMinX) & ! [ conditional(PlayerX - 1 = BlockX) & conditional(PlayerY = BlockY) ].

event(MoveRight) - PlayerX(PlayerX + 1) :   
conditional(PlayerX < GridMaxX) & ! [ conditional(PlayerX + 1 = BlockX) & conditional(PlayerY = BlockY) ].

// Pushing the block.  
event(PushBlockUp) - PlayerY(PlayerY - 1), BlockY(BlockY - 1) :   
conditional(PlayerY > GridMinY) & [ conditional(PlayerY - 1 = BlockY) & conditional(PlayerX = BlockX) ] & conditional(BlockY > GridMinY) & conditional(PlayerHoldingShield = 0).

event(PushBlockDown) - PlayerY(PlayerY + 1), BlockY(BlockY + 1) :   
conditional(PlayerY < GridMaxY) & [ conditional(PlayerY + 1 = BlockY) & conditional(PlayerX = BlockX) ] & conditional(BlockY < GridMaxY) & conditional(PlayerHoldingShield = 0).

// Pickingup shield  
event(PickupShield) - PlayerHoldingShield(0) :   
conditional(PlayerX = ShieldPosX) & conditional(PlayerY = ShieldPosY).

### Issues and Solutions

This can obviously be improved. There are two big issues I see with this.

* There is a lot of repetition in the rules definition. Moving the player up or down is a very similar concept— so the similarities should be compressed and into it’s own reusable unit. The same can be said for the block. Making sure the player does not have the same position as the block should only need to be written once, but currently it’s spread across multiple events.
* You cannot bundle data. For example, adding a second block is an arduous and monotonous task. This is partially because of the first issue, but also because the data and rules for the block are not bundled together.

These are my next two big tasks: reduce the repetition of similar rules and allow for easier bundling and duplicating of data. I think this is off to a good start but there is still tons to do.

Tagged in [Game Design](https://medium.com/tag/game-design), [Game Development](https://medium.com/tag/game-development)

By [Ryan Rothweiler](https://medium.com/@ryanrothweiler) on [April 3, 2017](https://medium.com/p/e7f4fcf95899).

Exported from [Medium](https://medium.com) on June 22, 2017.