Using ResEdit to Enhance STORM 0.9B1

Jim McCord (CIS 76077,1047)
January 1, 1992 (Happy New Year!)

Juri Munkki's great "almost-finished" STORM 0.9B1 (found here in LIB2) can be substantially enhanced and customized using ResEdit. In particular, it is possible to enable additional playing fields that are not accessible in the released beta version, change the order in which these fields are encountered, and modify the parameters of the various enemies that are encountered at each level.

While we're waiting for the "final" version of STORM to appear, those of you who are comfortable with ResEdit may enjoy hacking the game to make it more complex and change the way it plays. This note provides some general guidance on how to go about it. It assumes that you know how to use ResEdit (including a knowledge of how thoroughly ResEdit can mess up your file — use a copy!) I haven't provided specific details because once you know where to look it's pretty clear what to do. (Also, I have only explored the consequences of changing a few of the values that can be modified — the others are left as an exercise for the reader!)

Juri says he has no objections to my posting this note, but it would be best if you do not upload versions of modified games that you create. Juri says it is almost certain that these modifications won't work with the final version of the game (due to be released as shareware within the next month or two under the name "Arashi"), and in any case the final version will include a "human-oriented" editor that will obviate the need for ResEdit hacking.

Juri also sent an working document called "STORM Guidelines" that discusses the various trade-offs inherent in configuring the game. This was prepared for the development team but is also interesting to would-be customizers. I have attached it as an appendix to this note. One of its main points is that "more is not always better," and that elegant simplicity is preferable to quantity. This is worth taking to heart as you modify the levels and parameters.

Adding Levels and Playing Fields

Use ResEdit to open the game and double-click on the icon called "LEVL." This will produce a list of level IDs in which you will do all of your editing. Each ID corresponds to a level in the game. Double-clicking on any one of them will produce a list of the parameters that can be modified. Attempting to put a LEVL window away produces an "invalid rectangle" message which can be ignored. (Answer "yes.")

The first important parameter is "lvField" which defines the shape of the playing field for that level. The released game enables 8 of these — 100, 101, 103, 102, 104, 110, 119 and 112 in the order they are encountered (corresponding to level IDs 128 through 135.) I have found nine additional working fields — 105, 106, 107, 108, 109, 111, 113, 114 and 115. As I recall, 116 and 117 don't exist and 118 produces a diagram but freezes the game when you select it. I haven't looked to see if anything exists beyond 119.

Additional playing fields can either substitute for a field in an existing level (just replace the number) or you can add levels. To add a level to the game, duplicate one of the LEVL IDs, open it up, and type the number of the play field you want to use. If you want a 1-1 mapping of fields to levels you can have up to 17 levels. If you're willing to duplicate playing fields you can have as many levels as you want (subject, I suppose, to memory limitations) with different enemy parameters.

The other important parameter here is "lvNext." This is a pointer to the LEVL ID that will come up after you've conquered the current one. If there is no pointer to a given LEVL it will not appear in the scrolling selection list and you won't have access to it. (In the released game this is case for LEVL 1000, which implements the basic circle field with zillions of aggressive enemies.) The pointer sequence determines the order in which the levels appear, so you can make your own judgments about which play fields are easiest and which are hardest. For what it's worth, my own assessment in order of increasing difficulty is 100, 103, 113, 101, 114, 115, 104, 102, 107, 119, 108, 109, 105, 106, 112, 110 and 111. (Note, however, that the fields do not need to be ordered in increasing level of difficulty — it might be reasonable to have a simple field appearing late in the game with an aggressive set of enemy parameters.)

The other LEVL parameters are less important. "lvBonus" is self-explanatory. "lvColor" defines the level color (128 is blue, 1000 is red — other values may produce different colors.) "plSuperZaps" is the number of SuperZaps per level, and "shSpeed" and "shPower" defines the speed of each shot and the effect of each shot against spikes (higher values blast more of the spike.)

Enemy Parameters

STORM has four basic types of enemies — flippers, pulsars, fuseballs, and spikes (which, along with the "spiker" balls that form at the top of spikes and the "spiker plasma" balls that are hurled upward by newly-forming spikes, form the defensive armament of the enemy.) In addition, flippers, pulsars and fuseballs have "tankers" (the colored squares) that carry copies of the enemy for which they "tanker" and which dispense their basic types when you shoot them or when they reach the outer ring of the play field. (For what it's worth, it's a mistake to shoot pulsar tankers!)

Parameters for each of these threats can be edited. Some of the parameters are the same for each type (such as the points for killing one, how many can be "born" for each level, etc.) and some are unique (such as "flRot" which defines the speed at which the flippers rotate across segments.) You can deduce which parameters apply to which enemy by reading the parameter names. Juri's Guidelines document will further clarify this and provides guidance on reasonable parameter values. The "boredomCount" and "boredProb" values are particularly important and are discussed in some detail in the Guidelines. The rest you can probably figure out for yourselves.

In order to simplify your job of editing the parameters of each level I have included an Excel spreadsheet with most of the important parameters typed in. (Add the others if you want.) I've included my own values for some of the parameters, but don't take these as gospel — experiment to see what values work best for you. Once the final version of the game is released (along with its editor) I hope that people will post parameter values that they think work well.

Wrapping Up

After you're finished editing, save the game and start it up. If you did things right the play fields you elected to include will appear in the scrolling list at the bottom of the screen. Blast away!

APPENDIX

STORM Guidelines

((c) Juri Munkki and the STORM team, please do not modify)

Introduction

The first thing that you notice about STORM is that it has flashy 3D graphics and two channels of digitized sounds. The animation progresses smoothly and you can easily relate sounds to graphical actions. The STORM graphics theme is that of rotation and scaling. These very simple 2D geometric operation can be used to make the game look three-dimensional.

Special effects alone do not make the game playable. The balance and timing of the game are more important factors than what the game looks like. A very good game can be designed with very poor programming skills if the designer just makes the right decisions and has good ideas. In our case, we borrowed the ideas and experimented as much as possible to balance the game nicely.

Since this document is written at a time when the editor is still not ready (although it should be), internal STORM units are used instead of those that will appear in the editor. In most cases, it's a question of a linear transformation. While the editor is not available, changes to levels can be made with ResEdit. A template resource is provided with the game.

Things You Shouldn't Change

Some STORM constants are hard-coded into the program using #define commands in the source code. In theory you could change these values, but we are confident that the current set of values represent a working solution.

To provide the required perspective effect, certain units of measurement had to be defined. These units bear no relation to real-world units, since they do not change as the screen resolution and size change.

The distance from the player to the display was defined to be 19 units. You could think of the units as inches. If you do, you could say that the screen is assumed to be 19 inches (about 50 cm) from the player's nose. The top of the grid is assumed to be located on the same level with the screen. The bottom of the grid is defined at 120 units from the screen. This means that it's about 3 meters away.

All vertical speeds are defined as units/round. A unit was defined above and there are about 20 game rounds per second. A speed of 1 vertical speed unit could then be thought of as 20 inches per second. Typically a flipper advances at just about that speed, but since fixed point numbers are used for flipper speed and distance, you can go much slower or faster than this. Use 1 vertical speed unit as a base for your exploration.

Rotation is defined with the angle unit. This is actually something that could also be changed with a single change in an include file, but you would end up messing up our whole measurement system. There are 120 angle units in a full circle, so a single angle unit is equal to three degrees.

Angle units are normally integers values and angular velocity is defined in the same way as vertical speed. A safe angular velocity would be ±3 units per round or in other words 180 degrees per second. A full revolution at that speed would take two seconds.

Our tests showed that this angular resolution was sufficient and that fixed point numbers were not necessary to represent angles. You are welcome to use fixed point notation, if more accuracy is desired.

Other movement directions are harder to describe. Usually things are measured in fractions of the width of a grid lane. A fuseball that is traveling at 0.2 lanes per round will move the distance of four lanes (or segments) per second. If the typical grid is made of 16 segments, it takes 4 seconds for the fuseball to go around that grid. The current implementation of STORM doesn't allow you to edit any of the speeds that are related to segment size. Since only fuseballs use these and their movement patterns replicate brownian motion (with a speed limit), you shouldn't find it necessary to edit that value.

In designing the levels we found that 16 segments are ideal. The minimum number of segments should probably be 12 and the maximum is likely to be about 22-24, depending on the grid shape.

Things You Can Change

As shortly mentioned above, you can change the advance speed and rotation speed of enemies. Time units are always defined as rounds, with 20 units being one second. For instance, if you define that a pulsar pulsing time is 20 units, a pulsar will wait for one second before flipping. You can start out with a value close to 10 when experimenting with timers.

The most important thing related to playability is the probability at which the enemies are created. All enemies should share a common boredom factor. This factor is added to their base probability of creation when the playfield has less than a defined number of enemies.

Basically, for each enemy type, a random number is picked 20 times per second. This number is between 0 and 65535. This number is compared with the individual probability of that enemy type. If the total number of enemies is smaller than the required number, the boredom factor is added to the individual probability to increase the odds of a new enemy.

When a new level starts, the number of enemies is usually 0. For the few following rounds, the boredom factor is used to quickly create an army. After these are created, the boredom factor has no effect unless the player successfully destroys enemies. It is of course possible and desirable that new enemies are created even if the required minimum number is already present. In extreme cases, the game might seem unfair, but then again, life ain't fair.

There are a few things to watch when defining the boredom count and probability. If the count is too low and the individual counts aren't high either, you'll end up with nearly empty playfield. A good player will have no problems clearing up a level like this.

If the individual counts are high and you are not using the boredom count, you'll end up with a level that acts highly randomly. A good player might have serious problems with a level like this, while a lucky beginner could win with no problems.

Obviously, a good compromise is required. Our 0.9A-series of levels consisted of 7 different difficulty levels. Here's a table of flipper probability levels and boredom counts and probabilities for all those levels:

Level	*Flipper P*	*Count*	*Boredom P*
1	768	2	4096
2	1536	3	2048
3	1408	4	2048
4	1792	4	2048
5	2048	5	2048
6	2048	6	2560
7	2048	8	2560

Lowering the count value has a much larger effect than changing the probability. Our tests show that the above values work well for the lower levels (1 through 4), but tend to be a little high for the more advanced levels. The problem isn't that the game is too hard to play, but that all the enemies are created almost at once and the player can just zap them away with the superzapper.

A better guideline would be to keep individual probabilities around 1000-1500 and slightly increase the boredom probability and count as levels become more advanced.

The amount of enemies per level should increase faster than it does in our test case, where level 1 had 6 flippers and the last level had 7 flippers and level 8 had a total of 23 enemies (counting tankers as a single enemy). If the total count of enemies is too low, the boredom count dominates the game play. Once the boredom count is reached, all the player has to do is to destroy all the enemies.

We haven't yet found the optimal set of values, but the 0.9A-series values have proved to be quite entertaining. Janne Frösen is capable of playing all eight levels without loosing a single life, so we concluded that the levels need more enemies and less importance on the single superzapper. We also feel that the amount of superzappers should not be increased from 1 per level, although it might be desirable, if the level is intended to be extremely long and difficult. A better solution would be to split the level into two levels with one superzapper per level.

Spikes and spikers are handled differently than other enemies. A spiker probability of 0 creates no spikers and in that way doesn't allow spikes to grow any larger than their initial size. A good spike probability to try is about 100.

Since spike depth is measured in depth units (as described in earlier), the values can currently be between 0 and 120. Although these are the legal bounds, we recommend that the values be kept between 5 and 120 instead. If you are designing an advanced level with no pulsars, 60 is a good starting value. If there are pulsars, you should probably increase this value to make the spikes shorter. Long spikes make young pulsars especially dangerous.

Player shot power is only used as a measurement of how much a spike drops when it is hit. This is most important during fly-through, where it is sometimes necessary to shoot one's way through the spikes. A shot power of 2 seems to work extremely well on more advanced levels.

All current levels use a shot speed of 4 units/round, so unless you wish to make a dramatically different level, we suggest that you do not change this value. Higher values make the game easier, while smaller values make it harder. This is derived from the fact that the player can only have a fixed number of missiles active at a certain time. If the shots travel fast, the player doesn't run out of shots quite as often. If the shots are slow, they are grouped more tightly and better timing and skill are required from the player.

Should It Pulse, Fuse or Flip?

It is quite easy to control the feel of the game by creating different mix ratios of enemies. Since you have five enemy types available there are obviously a lot of choices to be made.

In the classic Tempest video game, the game starts with only flippers, then adds flipper tankers and spikers pretty soon and later on advances to fuseballs, pulsars and tankers of these types. This represents just one way to arrange the way that enemy types are introduced, but it is a fairly good way, since flippers tend to be easier to cope with than the other types.

Each enemy type has a different effect on how the player can win the level. What this means is that when a player for the first time reaches a level with a new enemy type, he/she will probably have problems advancing to the next level. After a few trials and errors, the player will usually learn a workable scheme that will improve as he/she gets more experienced.

Flippers are the most basic enemy type. Slow flippers are very easy to handle and can usually be destroyed before they reach the edge of the playing grid. More advanced flippers force the player to seek a relatively static position. This position is always in a spot where flippers have to rotate a wider angle from one lane to another. The advantage to the player is that flippers can be shot while they are rotating onto the player's lane.

Pulsars are close relatives to flippers. Pulsars are much more dangerous than flippers because they do not have to reach the edge before becoming lethal. After each rotation from a lane to another, a pulsar stops rotating for a while and just "pulses". While a pulsar is pulsing on a certain lane, the player can not move onto that lane without getting killed. For this reason, having pulsars on a certain playfield tends to limit the movement of the player. On a playfield where there are only pulsars, the player will usually seek the same kind of position as shown above.

Fuseballs are quite different from pulsars and flippers. They tend to drift midway between the bottom and top of the grid. Sometimes a pulsar leaps up or down. These movements are random, but if the player is on the same lane with the fuseball, the fuseball is more likely to leap in some direction.

If the fuseball leaps down (away from the player), it will appear to be trying to evade player shots. If it leaps up, it looks like it has decided to attack the player. For this reason, fuseballs are quite effective against a static playing style. Fuseballs are highly recommended in order to force the player into moving around the playfield.

Fuseballs are especially dangerous when they reach the edge, since they can only be destroyed with the superzapper. For this reason, the amount of fuseballs on a certain level should be related to the amount of superzappers that are allowed on that level.

Spikes, spikers and spiker plasma balls are all related. They make up the whole defensive side of the enemy. A spike protects other types of enemies from getting hit. As was mentioned earlier, you shouldn't start with extremely long spikes, since they tend to make pulsars particularly nasty.

Tankers are a surprisingly varied type considering that they all act the same with the exception of splitting into different enemy types. The flipper tanker is just an easy way to quickly transport flippers to the edge. A tanker is more likely to succeed in bringing a flipper to the edge since it doesn't flip or move sideways. A static player is less likely to score a hit on a tanker.

Pulsar tankers are interesting in that destroying one actually makes the game harder. This is because the player can not move in any direction after destroying a pulsar tanker. The best technique is to avoid shooting pulsar tankers.

Fuseball tankers are easier, if they are destroyed very early. If a fuseball tanker reaches the edge and the player has already used all superzappers, the chances for survival are pretty slim.

For a particularly nasty level, you might consider creating fuseball and pulsar tankers with the same color. Some players may feel that this is not a fair technique, so don't overdo it.

These are only preliminary guidelines and somewhat incomplete. Hopefully we will some day have an editor so that the it will be easier to experiment with different level layouts.

---------- Footnotes ----------

1. At this time , the game levels still need a lot of editing and only 8 levels are available.
2. The current maximum number is 9. We experimented with 8, but 9 was found to be better.
3. Actually you have seven types, if you count each tanker type separately.

---------- Sidebars ----------

Created: Monday, April 22, 1991 Last change: Wednesday, April 24, 1991

Project STORM Author: Juri Munkki
Guidelines

Flashy graphics and groovy sounds aren't enough to make a game a big hit. STORM allows users to create their own games and edit existing ones, so some guidelines are useful to have around.