Synergy

Lead Designer
Andrew 'Zif' Horton

2d/3d Art & Animation
Matthew Giacomazzo

Lead Programmer & Assistant Designer
Brian Ratta

UI Artist & Additional Animation
Andrew 'Zif' Horton

Tool Programmer
Andrew 'Zif' Horton

Audio FX & Original Music
Matthew Giacomazzo

Voice Talent
Sera Galvin

I. Artist Statement / Philosophy / The WHY Factor (why create this game? why would someone want to play it?)

This game is truly an experiment in cooperative gaming, through this game we hope to demonstrate that it is possible for people to work together in a non split screen environment, and that it is possible to have a great time even when in a supporting role. People will enjoy playing this game for a variety of reasons, but it's greatest thrill could be counting on someone else, and not knowing for sure what will happen because you are trusting someone else to a part of the task instead of doing it all yourself.

II. Predecessor or previous games/ distinctive factors in this genre

There are several games that have incorporated some of our ides for this project. Just as our game title implies multiple people working together we have combined ideas from other games and our own to create something new.

Game	Feature Description	Difference
Ikaruga	Ikaruga is a traditional bottom up scrolling space shooter game with one twist absorbing and firing black and white weapons. This feature is very similar to our implementation of the color system in red, blue, green, and yellow. Absorbing a color allows the user to release a special attack which is the same for both colors.	In our game the different colors also cause the attacks of the character to be inherently different, the play is done in solid areas instead of a scrolling interface, and attacks do not mean certain death but rather damage. We also have special attacks charged by absorbing attacks but each special for the each color will have a different effect.
DDR (Dance Dance Revolution)	This is a game in which players must quickly press arrows in sync with music. It requires excellent timing and has a unique skill set that develops of eye-foot coordination. In our project we are using the same arrow layout and skills in quickly reading arrows and performing that task with ones feet will be essential. Also the rush of physical activity is very important in this game, as it will be in ours as well.	Our game may use arrows, but they are for the most part not laid out the same way every time as in DDR. Also timing is less important than speed and accuracy. The timing in our game is not an arrow lining up but rather reactions to the gameplay and required color changes.

III. Target Audience

Our primary audience is more or less the social gamer. As this is a cooperative game people who like to play alone or compete alone would probably not enjoy the game. As there is a slight consideration of people having a game pad available our main audience will most likely include people who already have a pad because of another game which uses it. However if word of mouth gets around that the game is fun on it's own merits this would no longer be as much of an issue. In a more far thinking sense if a game of this sort were successful we could even sell our own custom pads got better control schemes and graphics to go with the theme. A secondary audience for this game could be those interested in a fun way to exercise. Due to the fast paced feet motion of the game it is excellent at creating full body aerobic exercise.

IV. Introduction & Story

The game is not really story driven so much as action driven. The introduction will be more an introduction to the rules of the game and how to play. Eventually we may create a "Story Mode" which fleshes out the character of our sorceress and her familiar but currently we are focusing on the game play itself. This could include the character and monsters changing over time as well as an inventory and puzzle system.

V. Immediate and long term projected socio/cultural project impact

Games have become in many cases almost an anti-social activity. We hope our game will greatly encourage direct interaction and relationships with other people through the game's unique interface. In the long run this kind of close interaction could stand to encourage the players to work together more fluidly and quickly. Also it may encourage the delegation and assembly of separate tasks to accomplish a greater goal, a lesson that must be learned in order to be successful in the game. Finally this could forward the idea of games that are actually healthy to play, a win-win situation for everyone.

VI. Delivery System & Requirements

To package our program we are using Inno Installer. This is a free installer that is easy to use and looks very professional. Delivery can be done in two ways. One as a burned CD with our setup and install program. Second is a website where people can download our program. The requirements to run our game will include the OpenGL and glut DLLs. Our game provides these on Install.

VII. Interface

Type	**Implementation**
Primary movement	W,A,S,D keys control strafing of the character
Turning	The character points in the direction of the cursor, but always floats as per the primary movement (S is always straight down and so on)
Fire Primary Weapon	Left mouse click fires currently selected weapon (the current player color)
Fire Special Weapon	Ctrl-clicking causes a stored special attack to be fired, these attacks are fired in the order they were charged.
Target Weapon	Weapon will fire directly at wherever the cursor is pointing
Change active color	Hitting corner arrows in a jump (up, left; up, right; down, left; down right) will change the active color
Charge special attack	When in Charge mode, the first arrow a player hits will determine what attack will be charged (as indicated by the first block that shows up) immediately upon hitting that arrow the remaining arrows move towards the top of the screen disappearing when hit on the pad. Upon completing each segment one charge is added. Magic power drains over time and the mode will automatically quit if there is no more remaining.
Activate Special Attack	Right clicking activates special attack mode which drains magical power over time to be in.
Change Combat Mode	Jumping on the left and right arrows at the same time then a direction changes the battle mode of the player. These modes are Increased Speed, Damage, and Defense.
Quick dodge	Tapping an arrow twice in the same direction quickly causes the character to teleport in that direction in order to dodge attacks.
Enter Portal	Hitting Enter while over a portal takes you to another area.

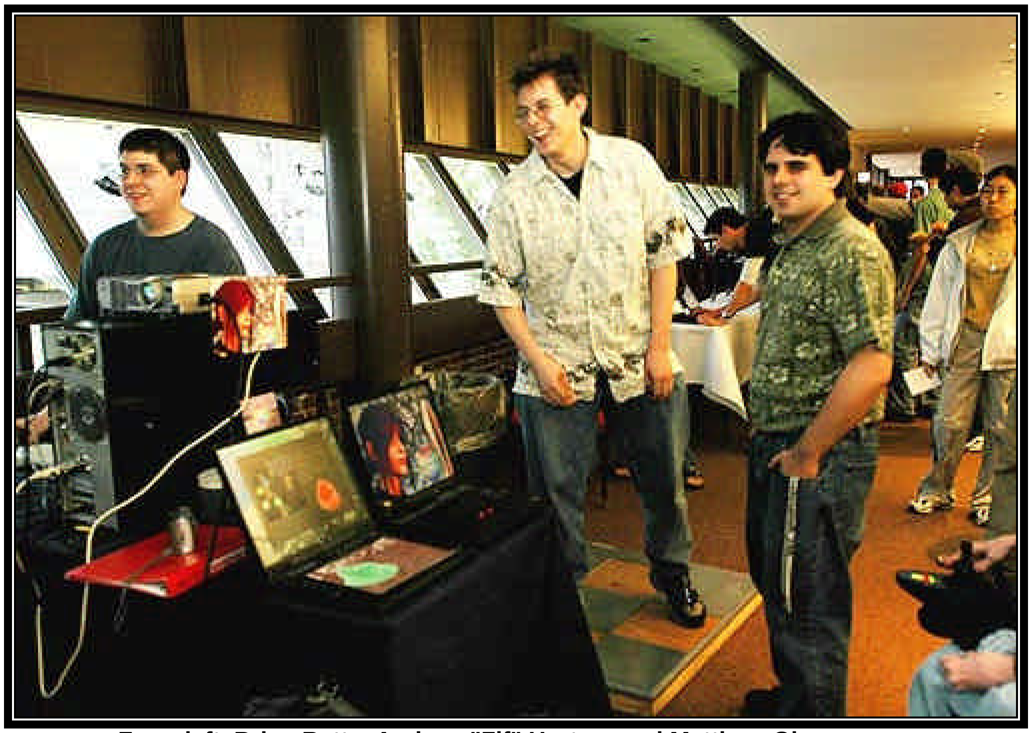
VIII. User Interaction

Key/Mouse Player-> Game events

The Key/Mouse player will be constantly interacting with the various enemies and attacks on the screen. Commonly both dodging and firing, even running into same color attacks to increase magic charge.

Pad Player -> Game events

The pad player must keep changing the current active color to what is most applicable to the situation at hand, including absorbing a particular color or using a color's attack. Also they must perform arrow combinations displayed for charging special attacks, Dodge attacks with the speed dodge feature, or change the combat mode when appropriate.


From left, Brian Ratta, Andrew "Zif" Horton, and Matthew Giacomazzo
Photo Credit: Luanne M. Ferris – Times Union

Player->Player

The Key/Mouse player and the Pad player will interact with each other. This could take the form of direct communication, the pad player saying "red" to turn red for example, automatic communication, the pad player changing the color on their own to one more convenient and the main player responding to this for example. And finally synergistic interaction wherein the players having gotten used to what the other will do respond to things exactly when they need to happen. The main player also activates and deactivates charge mode while the pad player chooses what color special to charge and how fast they are charged. This is one of the most interesting and challenging features of the game, allowing experienced teams to gain an edge by knowing one another well.

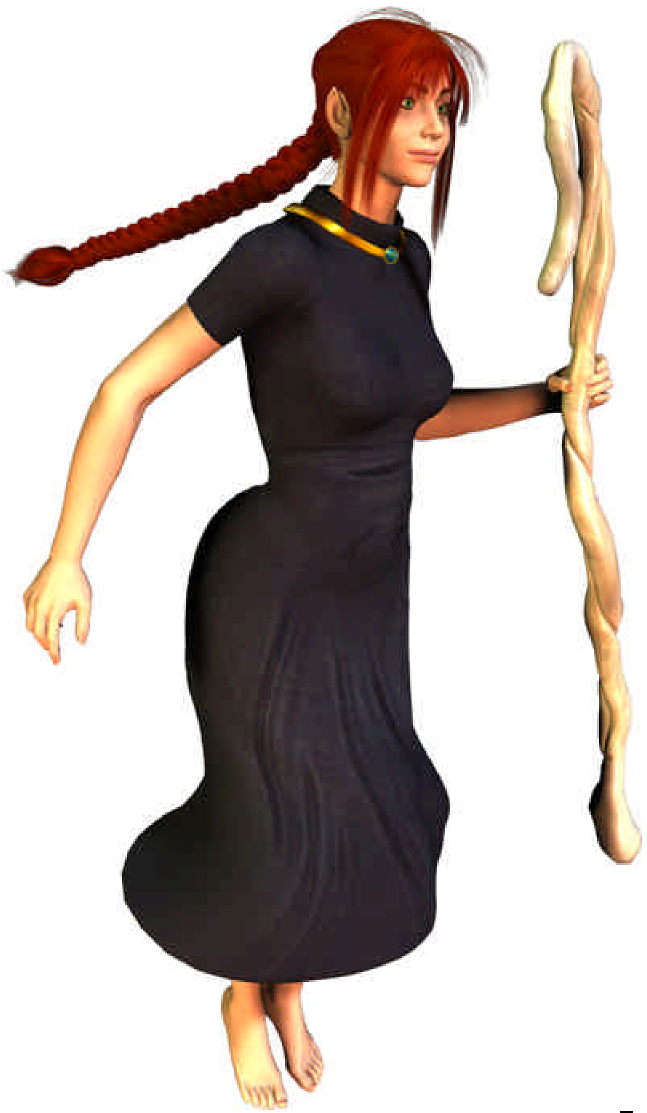
IX. The World Layout

The world setting is high fantasy. Expect to see plenty of trees, grass walkways, stone paths and walls, and interesting decorations scattered throughout. Monsters and color towers dot the landscape challenging players who enter.

X. Level Design

Currently levels are created using ASCII tile maps which are created using the synergy Map Editor. Monsters walls and other objects can also be added easily using this editor. As the game develops so will the level designs, currently they consist of some number of areas connected by portals which are activated by pressing enter. By traversing these portals players enter new areas with monsters which respawn each time. Eventually these will be replaced with more complex areas including features such as shops, items, bosses, and full level switches.

XI. Visualization- characters, flow charts,

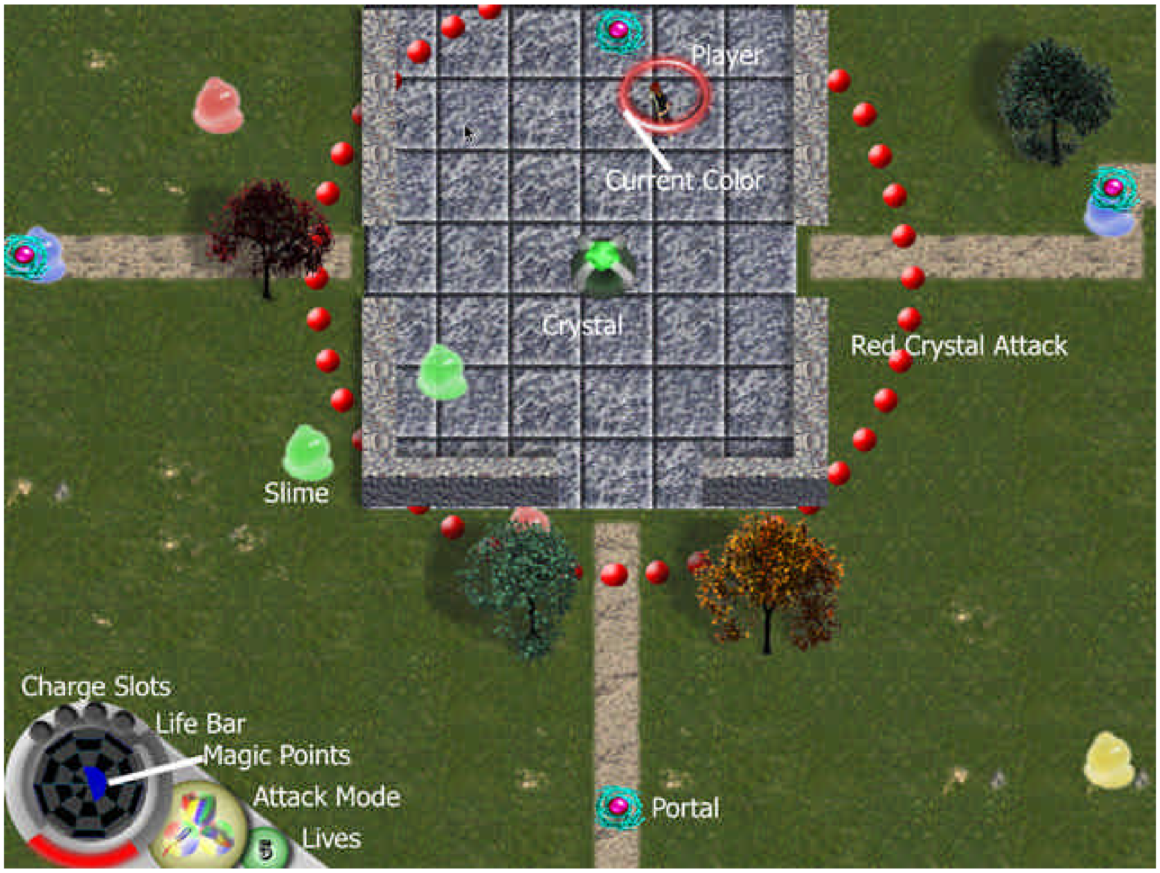


Characters

1. A sorceress rendered first in poser then taken down to sprites, fitting within a 128 pixel tall area (leaving room for the familiar as well)
2. A familiar, fairylike that flies around the main character leaving a streak of the currently active color, a ring rendered in Maya then influenced to give a perception of rotation.
3. Slimes, The base enemy of the game. Slimes come in all 4 colors and cycle between 3 different AI's. They can either hold their ground, charge, or fire off attacks.
4. Fire Monsters, very aggressive but short lived fireballs that seek out the player
5. Wind Monsters, a dangerous quick moving tornado

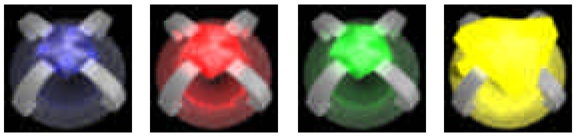
HUD

1. The health bar is the half circle along the bottom of the HUD ring
2. Magic level is represented in the center of the HUD by the concentric circle gauge.
3. Colored spheres along the top of the HUD represent the charged special attacks.
4. The right side features arrows when in charge mode that must be entered to charge up.



Enemies

There are two kinds of enemies presently, the first kind we call crystals, which are featured in the corners of the screenshot above. Every 8 seconds they fire out a large volume of predictable attacks in a quarter circle pattern. Hitting them with an attack causes this timer to reset.



The Second kind of enemy is more dynamic; currently in the game as only white circles they will soon have an animation and an AI that will allow them to attack the player. A level will end when all of this kind of enemy has been defeated.

XII: Music/Sound Design

Sound Effect Production and Format Specs:

Sound effects will be produced and mixed from original recorded sources and free royalty-free sound libraries. The final mastered sound effects will be converted down to at most 8bit 22,050hz sound samples in the standard WAV format. The reason for the slightly lower sound quality, about equal to radio, is that this is common practice to save space and put less strain on the game.

Music Production and Format Specs:

Original music will be produced through a variety of hardware and software applications eventually leading to a final mix-down in Cakewalk Pro Studio 9 and mastering in Sound Forge to a final 16bit 44.1khz WAV format file of CD quality. This will then be converted down to 160kbps MP3s to save space. The music will be original and written fresh for this game to help add to the mood and enhance game play.

Audio Theme/Mood Objective:

The theme of the game is high fantasy so the audio will be made to match that. Music will be big, bold, epic much like a movie soundtrack with use of multiple instrument sounds including drums, strings, horns and flute. Sound effects will be made to best match the actions be seen. Repetitious actions will have multiple sound variations as well. The large special attacks and character damage will also made using recorded voice talent.

Music List:

Song	***File Name***	***Length***
Main Gameplay Music	Matthew Giacomazzo - Synergy - Heart of the Adventurer.mp3	~4-6 minutes
Long gameplay track and main theme of the game.		
Powerup Gameplay Music	Matthew Giacomazzo - Synergy - Rising Promise.mp3	~45 seconds
A more upbeat tension-building verse.		
Opening Theme	Matthew Giacomazzo - Synergy - Opening.mp3	short
A short opening for the title screen.		
You Lose Music	Matthew Giacomazzo - Synergy - Death.mp3	short
Sad mournful music.		
You Win Music	Matthew Giacomazzo - Synergy - Level Complete.mp3	short
A short fanfare and recognition of accomplishment.		

Sound Effect List:

```
Use imp-lightning.wav for the impact sounds of each of the projectiles of the
lightning special.

Use the imp-fire.wav for the fire orb's melee attack

absorb.wav              - absorbing of attack
att-earth01.wav         - 3 versions of earth attack launch
att-earth02.wav
att-earth03.wav
att-fire01.wav          - 3 versions of fire attack launch
att-fire02.wav
```

att-fire03.wav	
att-lightning01.wav	- 3 versions of lightning attack launch
att-lightning02.wav	
att-lightning03.wav	
att-slime01.wav	- 3 versions of slime attack launch
att-slime02.wav	
att-slime03.wav	
att-superearth.wav	- Earth special attack
att-superfire.wav	- Fire special attack
att-superlightning.wav	- Lightning special attack
att-superlwater.wav	- Water special attack
att-water01.wav	- 3 versions of water attack launch
att-water02.wav	
att-water03.wav	
cancel.wav	- cancel button
chardeath1.wav	- 3 versions of character death
chardeath2.wav	
chardeath3.wav	
charhit1.wav	- 4 versions of character taking damage
charhit2.wav	
charhit3.wav	
charhit4.wav	
imp-earth.wav	- earth attack impact
imp-fire.wav	- fire attack impact
imp-lightning.wav	- lightning attack impact
imp-slime01.wav	- 3 versions of slime attack impacts
imp-slime02.wav	
imp-slime03.wav	
imp-superearth.wav	- earth special impact
imp-superfire.wav	- fire special impact
imp-superwater.wav	- water special impact
imp-water.wav	- water attack impact
imp-windbuff.wav	- wind monster melee
ok.wav	- ok button
ting.wav	- attack no effect

h

XIII. Rules and Gameplay A. Setup, B. Gameplay, C. Scoring

The Basic Rules

1. A player if hit by the active color absorbs magic energy equivalent to the power of that attack
2. A player hit by any other color takes normal damage
3. A monster hit by its own color takes no damage; special effects like freezing still occur.
4. A monster hit by the opposite color takes double damage.
5. Special attacks are performed by activating a special attack mode; every second in this mode drains a portion of your magical energy bar.
6. The pad player may charge any number of special attacks before they have to leave special attack charging mode.
7. During spell charging time it is not possible to switch colors, change attack modes, or speed dodge.
8. Special attacks must be fired in the same order they were charged, this is represented by a gauge in the lower left corner.
9. Crystal attack squares will send out an arc of attacks across the screen, shooting one of these will reset its counter before it fires.

Gameplay

Gameplay will be achieved through the application of the above rules. For some examples let us refer to the player using keyboard and mouse as A, and the player on the pad as B. A will constantly be dodging attacks that are not the current color and trying to absorb attacks that are of the color. Mean while player B will be trying to make this task more possible by switching to colors that have to be passed through.

Scoring (When Implemented)

<u>Score Increase Events</u>

- Enemy Destroyed
- Enemy Destroyed with opposite color (more points)
- Enemy Destroyed with same color (less points)
- Attack absorbed (small amount of points)
- Level Cleared quickly (point scale based on time to clear)
- Total enemies destroyed (point scale based on amount cleared in level)
- Multiple Enemies Destroyed (bonus points for killing more than one enemy in an attack, also keeps track of how many the most was for end bonus)
- Lives remaining (bonus points for not using all lives to get to the end)

<u>Score Decrease Events</u>

- Used a continue, remove all points

XIV. Program Structure

We have created a hierarchical structure starting from a top level class for our game. The main class at the top handles most of the gameplay including detecting collision and keeping track of the other objects. A level down there is an object class that all other objects will inherit from. Everything in the program will follow this same set of rules to make adding new objects a simple task requiring little re-coding. Other specific classes have been added to handle unique elements of gameplay such as the special attack charging interface.

XV. Technical Specs: Physics, Rendering System, Lighting Models

Rendering- All rendering will be done using a SDL interface. We will be pre-rendering sprites into bitmaps for the game to display. These will be created through a combined use of Photoshop, Maya, Poser, and Bryce.

Physics-We are using a simple 2D movement engine which takes into account velocity and acceleration. This will be demonstrated through monsters and the character colliding and bouncing off walls and each other knock back attacks, and the sliding movement of the main character.

Lighting- We aim for consistency in how the scene is lit by placing our light sources in the same place in the various rendering applications we are using, we have decided to place the light about 45 degrees up and to the left.

XVI. Implementation

While this concept could work in either a 2d or 3d interface in the interests of time we have decided to go with a 2d interface. We feel that this will not detract from the gameplay as it will not harm the cooperative aspect of the game. By designing the game to have more controls than one person can handle easily we hope to encourage a new level of cooperative gameplay. The pad system is also another way we are differentiating the two tasks to tie together this synergistic effect.

XVII. Production Timeframe/Version History

Version .1(finished)

- Small tile set created
- Open GL and glut implementation
- Bitmap support
- Basic Physics engine
- Main music track
- Stationary Enemies
- Basic Gameplay mechanics (color switching/weapon firing)

Version .2(finished)

- Improved tile set with non feature and feature tiles in many versions
- 2 kinds of road tile for each piece
- Improved Opening Music track
- Intro and spell charging music
- Full set of sound effects including alternate versions
- Full set of splash screens
- All new weapon animations
- Game start Menu
- Conversion to SDL image manager
- Transparent images implemented
- Art updated to include transparency, walls fully redone, trees no longer backed by grass
- Monsters with varying AI capability
- Lua scripting language implemented for level management
- Added configuration options for objects through lua scripting
- Full function map editor created
- Portals now link levels together
- Speed dodge feature
- Mode change feature
- HUD updated to include new features
- Fully playable as a game

Version .3 (in progress)

- Dialogue system with script interface
- Full shop system
- Items implemented
- Quick spells implemented
- Built in Joystick capability
- Various new tiles/monsters
- Boss creatures
- Discrete levels/quests
- Scoring/leveling
- Scripting system fully implemented
- Required map editor updates for new version
- Shot charge attacks

XVIII. Research

We looked into various game that have had cooperative gameplay and found that nearly all of them required that both players do the same function just through some kind of split screen function. Other examples of cooperative gameplay involve crafting on MMORPGs, or games where you control two separate vehicles or characters who sometimes work together. The closest style of gameplay I found was in a game called wakeboarding unleashed where one character could control a boat while the other skied behind it. Still our conclusion was that in general games do not require that players work directly together to influence just one character on the screen.

A second avenue of research is also now underway involving the physical qualities of an active game such as ours. Through research we hope to be able to maximize this quality while also maintaining interest in the game by players. Past game examples have shown that these qualities are by no means mutually exclusive and may make our game even more fun to play.

XIX. References

Open GL game programming, Kevin Hawkins, Dave Astle, Primatec
Open GL Programming Guide (red book)
A list of playstation 2 games with cooperative elements
    http://charon.sfsu.edu/corey/ps2coop/
A review of the best CO-op games on the market for ps2.
    http://ps2.ign.com/articles/427/427229p6.html

XX. Appendix

Synergy Map Editor Specifications

Goal:

The goal of this part of the project was to make a program that would make creating the map and script files for the game much easier. This involves creating a simple visual system that allows for the conversion from a visual representation of a map to the files of the map itself.

Definitions:

- Tile: An image from a tileset that when gridded out over a screen can create a full room/level. In the case of this game they are 64x64 pixels in size.
- Tileset: A set of tiles arranged in some number of rows and columns. Our tileset includes grass stone roads and walls.
- Object: like a tile but intended to be either animated or standalone. These are used for things such as level exits, creatures, and other such features.
- Canvas: The structure the map image is created in.
- Layer: The game requires that images stack in the right order, for our game that places them in this priority-> grass/stone/roads|walls|objects.

Required Features and Implementations:

1. Ability to begin from a random field of grass/stone
   Purpose: Grass is the base tile of the game, also the location of a particular tile of grass has no real use. Thus the requirement is in making the 12 tiles of grass create a field that does not look like it repeats. By having made 6 tiles without obvious features and 6 with them this is accomplished.
   Implementation: The code uses a random ticket system with 10 to 1 odds favoring the feature free tiles. These tiles are then placed in an array and displayed on the canvas. If stone is also selected or stone alone, it is simply added to the ticket selection.
2. Ability to place grass and stone individually with brushes.
   Purpose: Tiles especially stone need to be placed in particular slots, but still do not need to be selected indivually as to what piece of stone.
   Implementation: A bind callback detects left clicks on the canvas and executes the brush selected in the brush menu. It randomly selects a tile in that set then edits the grass array and updates the canvas image.
3. Place road individually and automatically select the correct road piece.
   Purpose: placing road tiles is tedious if done individually, automating it based on what tiles around the tile are also road tiles is a big help.
   Implementation: When you left click and the road brush is on the following events occur.
   a. The x/y tile is calculated
   b. The neighboring roads are calculated and stored using a binary notation. (one adds 8, one 4, one 2, and one 1).
   c. The result of this notation is the array value of the tile that should be placed there.
   d. The appropriate road is added to the canvas.
   e. Using binary compares the road allocation algorithm is run on any neighboring roads, making them match up to the new road. This is also executed when a road is removed.
4. Place walls with automatic correction as of roads, however on a new layer
   Purpose: walls have transparent areas so that they can be placed on top of other tiles and still have the background show through.
   Implementation: They are calculated using a similar function to the roads. Since the full array of tiles is created before the roads then updated walls will always be on top of tiles as desired.
5. Create Objects
   Purpose: To make creating and locating monsters, crystals, trees, and the player start location easy.
   Implementation: Objects being placed individually after the tiles and walls will always be on top of them. They are stored in growing arrays as they are made and popped onto the screen. These are located by the top left corner (because that is how the game itself locates them.
6. Open Map files
   Purpose: Being able to open and edit files you have already created
   Implementation: Made very simple because of the implementation of the random tiles one normally starts off. A very similar function is called that instead takes a completed array. This array is formed by parsing the opened map file with regular expressions.

7. Save Map files
   Purpose: Save files so they can be used by the game
   Implementation: This consists almost entirely of simply printing out the data structures used to store their locations for the program already. Grass, stone, and roads are stored in the top part of the file in a number:number / line arrangement, the walls are stored individually as they are a sparse matrix rather than a full one.
8. Save Script files
   Purpose: Save the scripts needed by the game for creating interactive elements
   Implementation: Print out the object data in the format defined by the script system. This changes on a per item basis see the section on script files for more details. The information itself is stored in the array as a Perl structure which is easy to iterate through. Most script items have the same information requirements, just arranged differently.

Script Engine Specs
Here is a list of all the features which can be in a script file, since these are not all implemented in the game yet some of them are also not featured in the map editor.
File Layout:

```
--start<space><new line>
<commands><new line>
$<new line>
--<event name><space><new line>
<commands><new line>
$<new line>
--<event name><space><new line>
<commands><new line>
$<new line>
!<eof>
```

Commands:
createTree(string ObjectName, int Frame, int Xposition, int Yposition)
    This creates a Tree object in the environment called ObjectName. The Frame num is the picture of the Tree to be shown. It is placed at position (Xposition, Yposition).

createSlime(string ObjectName, int Xposition, int Yposition, int Color)
    This creates a Slime object in the environment called ObjectName. It is placed at position (Xposition, Yposition). The Color is what color the slime will be.

createCrystal(string ObjectName, int type, int Xposition, int Yposition, int Color)
    This creates a Crystal object in the environment called ObjectName. It is placed at position (Xposition, Yposition). The Color is what color the crystal will start, the type is what direction it fires in.

move(string ObjectName, int Xposition, int Yposition)
    Moves an object that has already been created.

destroy(string ObjectName)

Destroys an object in environment.

callme(string EventName)

Calls a specific event that is loaded in the Virtual Machine. Usually only used internally, but can be used in the script file.

setTrig(string VariableName, string Relation, int Number, string EventName,string TriggerName)

Used to create a trigger that checks the given variable and number using Relation operation given.

setCollide(string ObjectName, string ObjectName, string EventName, string TriggerName)

NOTE*:* *TriggerName is the Trigger to remove after event has been done*

Similar to setTrig, it checks if the two given objects have collided. It then calls the event and removes the given trigger.

addList(string TriggerName, string Event)

This adds a trigger to the list of active triggers. Event should be a setTrig or setCollide that is in quotes. See Example script.

remove(string TriggerName)

This deactivates the given trigger.

turn-on(string TriggerName)

This reactivates the given trigger.

setExit(name ExitName, double Xposition, double Yposition, string FileToRun)

This creates an exit at given location

This script is basic. In my game code it calls events steve collide and print. Which creates trees Bob,Steve,and Joe and sets a collide trigger to Bob and Joe. When you run into Bob it calls collide2 which sets a trigger to Steve. When you hit Steven he is moved, and the same goes with Joe.

Example Script:

```
--steve
create("Bob", "tree", 100, 150)
move("Bob",234,200)
$
--collide
addList("test","setCollide(\"PLAYER\",\"Bob\",\"collide2\",\"test\" )" )
$
--collide2
addList("test2","setCollide(\"PLAYER\",\"Steve\",\"print2\",\"test2\" )" )
$
--print
create("Steve", "tree", 100, 200)
create("Joe", "tree", 100, 400)
addList("test3","setCollide(\"PLAYER\",\"Joe\",\"print3\",\"test3\" )" )
$
--print2
move("Steve",534,200)
$
--print3
move("Brian", "tree", 134, 500)
$
!
```

Program simple use instructions

To begin use **new file**, or **open file** from the **file menu**.
To use a brush simply select it in the brush menu then **left click**.
To erase a wall select the **erase** brush and **left click** on the wall.
To fill with a new background select if you want grass, stone, or both to be in the fill then choose **fill** from the **random fill** menu.
To place an object simply select it from the **object** menu and then **right click** where you would like it to be placed.
To see what the map file or script file will look like without saving it use **print map console** and **print script console** respectfully from the **file** menu.
To save the map or script file to a file use **save as** and **save script** from the **file** menu.

Install Requirements:
The synergy map editor requires that you have ActivePerl installed on a windows computer.
This can be downloaded from
http://www.activestate.com/Products/Download/Download.plex?id=ActivePerl
The code should run under Unix, it has not yet been tested however.
Once Active Perl is installed just extract the program to a directory and double click synedit.pl.