Trimuxs II Specifications

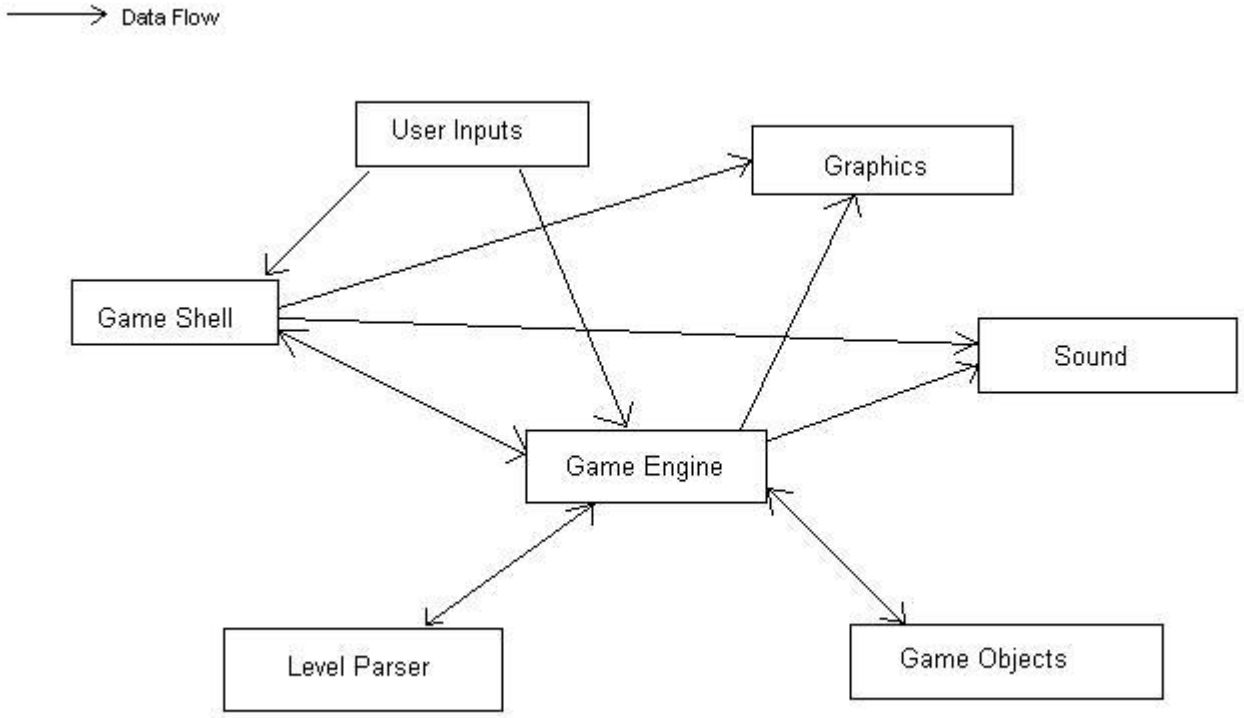
Alex Kossey

1 Project Description

Trimuxs II is a two-dimensional side-scrolling arcade shooter in the style of the Gradius, R-Type, or Thunder Force games. As someone who finds himself waiting around in the SunLab from time to time, I think that we could use a "high impact" game (at least, more so than Parsec) to help arcade addicts pass time.

Up to two players attempt (attempt being a key word) to save the galaxy from evil by flying through (perhaps) space, asteroid fields, or the stomachs of space worms. This involves blazing their ways through levels consisting of walls, various hostiles (mechanical or blob-oriented), a huge boss, and enemy shots (nondescript dots, colored lines, etc.), picking up cool power-ups (with names like "quantum disruptor") as the go in a never-ending attempt to survive frantic pace and large bosses.

2 System Model



3 System Components

The different components of the system are as follows:

3.1 Game Shell

The Game Shell is the basic loader for the game. It loads other components it needs (like graphics, sound, inputs, etc), displays the title screen, displays ending credits; basically it controls all aspects of the game aside from playing out a level. Users can set options like how many lives they have or what key or joypad buttons control their ships. When a player chooses to start playing, the Game Shell passes control to the Engine.

3.2 User Inputs

The User Inputs consists of all parts of the software that retrieve input from the user. Interprets keyboard and (ideally) joypad keystrokes and then sends this data to whomever asks for it, either the Game Shell or the Engine depending on what state the game is in.

3.3 Game Engine

The Game Engine controls the game when a level is playing. It loads the level from a file using the Level Parser and then runs through the level until the player(s) stop the game, are destroyed, or complete the level. The Engine would basically be a big loop (going through around thirty times a second) that, at each iteration, queries for user inputs, moves game objects like shots and enemies, sends information to the Grapics component to update what's on the screen, and checks for collisions. The Engine also plays sound effects and background music.

3.4 Graphics

The Graphics display graphics on the screen when they're told to do so by another component of the game (the Engine or the Game Shell). The component would interact with a library (likely SDL) to print what its given to the screen.

3.5 Sound

The Sound plays sound effects and music. It must be able to both play multiple sound effects and also loop music continuosly in the background. Like the Graphics, the Sound would use some library (once again, likely SDL) to play the required sound(s).

3.6 Level Parser

The Level Parser, given a file, loads information on the level from a file and then parses it into a form that the Engine can then use to play the level. Creating this "form" would involve producing all Game Objects (enemies, walls, etc.) that the level would need and making a data structure that

defines what events (enemies/walls appearing, sounds playing, etc.) occur at what times.

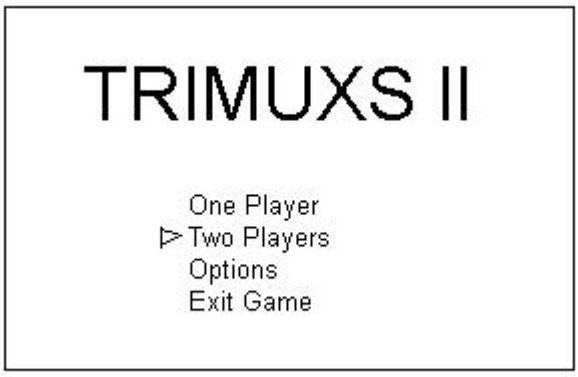
3.7 Game Objects

The Game Objects consist of the moving components that appear on the screen while a level is playing. This includes player ships, weapons, power-ups, enemies, walls, etc. These objects will all need to have functions that define their movement (either algorithms or user inputs), ways to represent their locations on the screen, size dimensions for collision detection, information about what happens when shots hit them, what kind of weapons they fire when, etc.

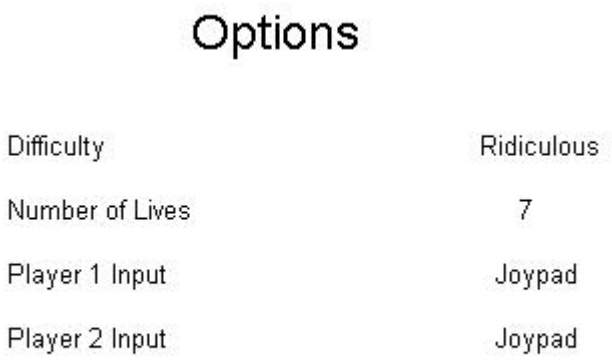
4 User Interface

The user interface will generally consist of two parts: the interface for the Game Shell and the interface for when the game is playing.

The interface for the Game Shell will consist mostly of a title screen and an options screen, which look like this:

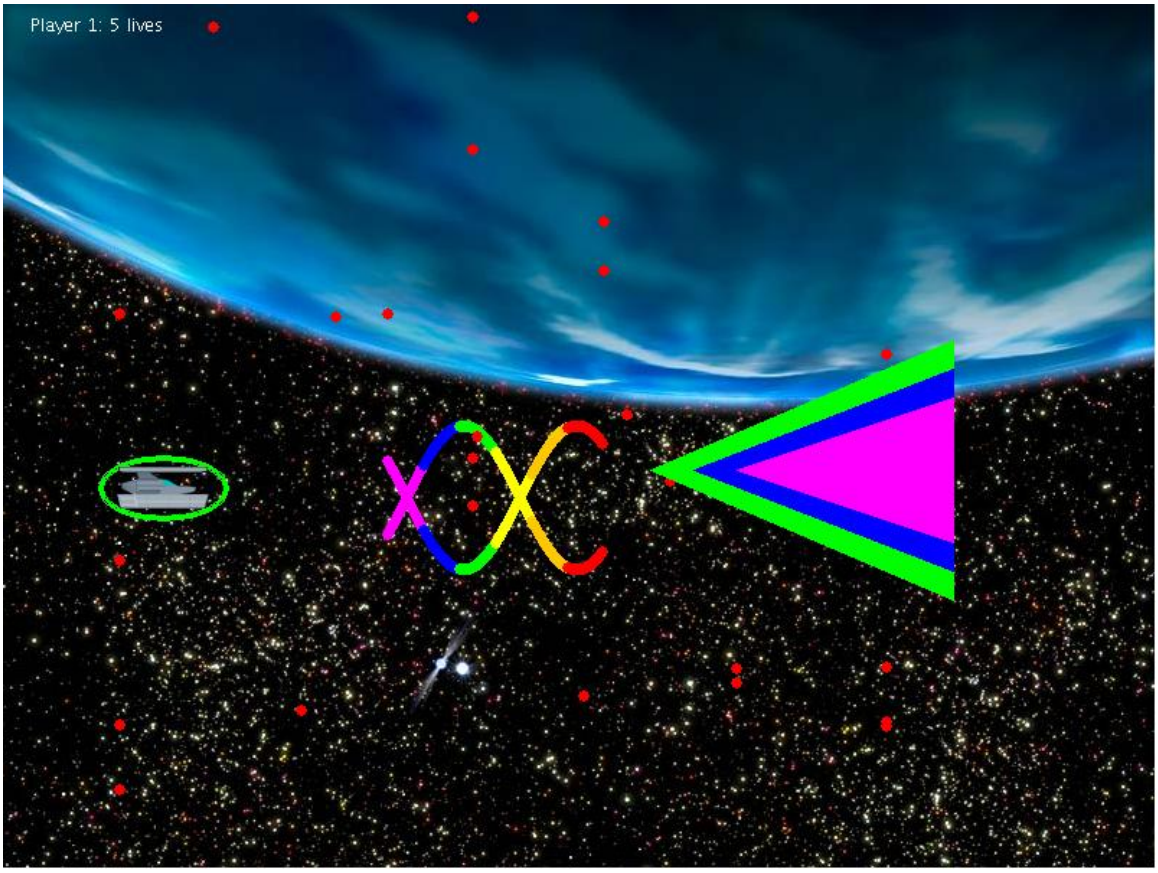


On this title screen, the players will be able to use an input device to choose to start the game with one or two players (which, if multiple ships are available, will lead them to a simple screen where they choose which ship to use), go to the options screen, or leave the game.



The options screen allows the user to set number of lives per continue, game difficulty, which keys on the keyboard correspond to different aspects of gameplay (moving, firing weaponry).

Here's a screenshot from Trimuxs I; Trimuxs II would have the same kind of look:



While a level is playing, the player can use movement keys (whatever they're assigned to be) to move the ship around the screen, fire weapons with the firing button, and pause the game. In addition to what's displayed here, the screen would ideally display what power-up(s) the players may have, as well as their current scores.

5 Other Requirements

5.1 Performance

The game must run reasonably smoothly (say at least thirty frames per second and ideally sixty frames per second) and avoid slowdown unless the screen is almost entirely full of enemies, in which slowdown can be an impressive effect (as in stage seven of Gradius Gaiden when the walls are collapsing all around the player).

5.2 Testing

All components described above are independently testable, meaning that each component should be tested comprehensively. At least a prototype engine would be required to begin attempting integration. Component testing would continue through integration. When the components are reasonably integrated, testing to make sure basic events happen at the right times could occur, followed by experimentation with border cases to try to catch all errors.

5.3 Reliability

The game should be able to run continuously from the time it's opened to the time to user(s) quit. Any errors with reading or playing the level files should be handled without making the program terminate. Once a level starts playing, it should be able to play through without errors.

5.4 Ease of Use

The game should be very easy to use, requiring only a very small number of keys to navigate through clearly-marked menus.

5.5 Portability

It would be nice if the software were portable relatively easily into Windows. Especially if we use a cross-platform graphics/sound library like SDL, this could be relatively easy.

5.6 Documentation

Documentation must be as extensive as documentation for video games usually are: it should contain information on the physical process of playing the game as well as "story" and information about a subset of the stages, enemies, and weapons present in the game.

5.7 Dependencies

The game requires a library to display graphics, play sounds, and handle user inputs. There are plenty of widely-used libraries that we could use.

SDL seems like an ideal option (especially since it does all three things and is cross-platform). Handling external hardware in the case of joystick support could be difficult and is certainly platform-dependent.

6 Requirements

Player:

- Move left, right, up, down, and diagonally (10)
- Fires a weapon forwards (10)
- Loses a life when colliding with an enemy shot or ship (10)
- Can pick up powerups that appear (10)
- If a player loses a life and still has lives left, a new ship belonging to that player must reappear without the game stopping play (10)
- Multiple kinds of ships to choose from (4)

Engine:

- Load levels and play them continuously until they are finished (10)
- Support both single player and two player modes (10)
- Must detect collisions, instruct game objects to move, etc. (10)
- Must be able to change the speed at which the screen scrolls (10)

Graphics:

- Rudamentary graphics on-par with old NES games (10)
- Able to display a background image (10)
- Credits screen that's not just text (4)
- More advanced graphics (4)
- Dramatic opening sequence (3)

Sound:

- Looping background music in some size-efficient format such as OGG or MP3 (10)
- Can play up to three WAV sound effects simultaneously (10)
- Different music for each stage (6)
- Boss battle music (4)
- Music for title screen, ending credits (3)

Inputs:

- Must be able to define inputs for up to two players on a keyboard (10)
- Joystick or joypad support (3)

Enemies:

- At least three types of enemy in every stage (10)
- Must be able to move in at least three patterns (10)
- Must be able to create shots that fire in a specific direction or in the direction of the player (10)
- Must explode when hit a certain number of times (9)
- Each level must have a boss at the end (9)
- Difficulty levels that define how many shots enemies fire or how fast they are (4)

Power-Ups and Weapons:

- There must be a basic player weapon (10)
- A forcefield power-up that allows the player to be hit multiple times before losing a life (9)
- Some kind of laser weapon (9)

- Different kinds of weapons available from powerups dropped by destroyed enemies (9)
- Missile weapons that fire in addition to the forward-shooting weapon (7)
- Weapons that seek out enemies or roll along walls (4)
- Some kind of appendages that attach to player ships to increase their firepower, either by giving them a higher rate of fire or changing the kind of weaponry they can use (4)
- A unique weapon for each type of player ship (3)

Levels:

- Must have at least two fully-functional levels that are noticeably different from each other (10)
- Must be stored in files in a relatively easy-to-define format like XML (10)

7 Possible Risks

Making such a game requires a sizable creative component: we would need a way to get graphics (music is more accessible, as I can do that). Regardless, creating artwork and sound and designing good levels and monsters takes a lot of time; it's hard to predict how long a particular idea might take to realize.