Trimuxs II Specifications

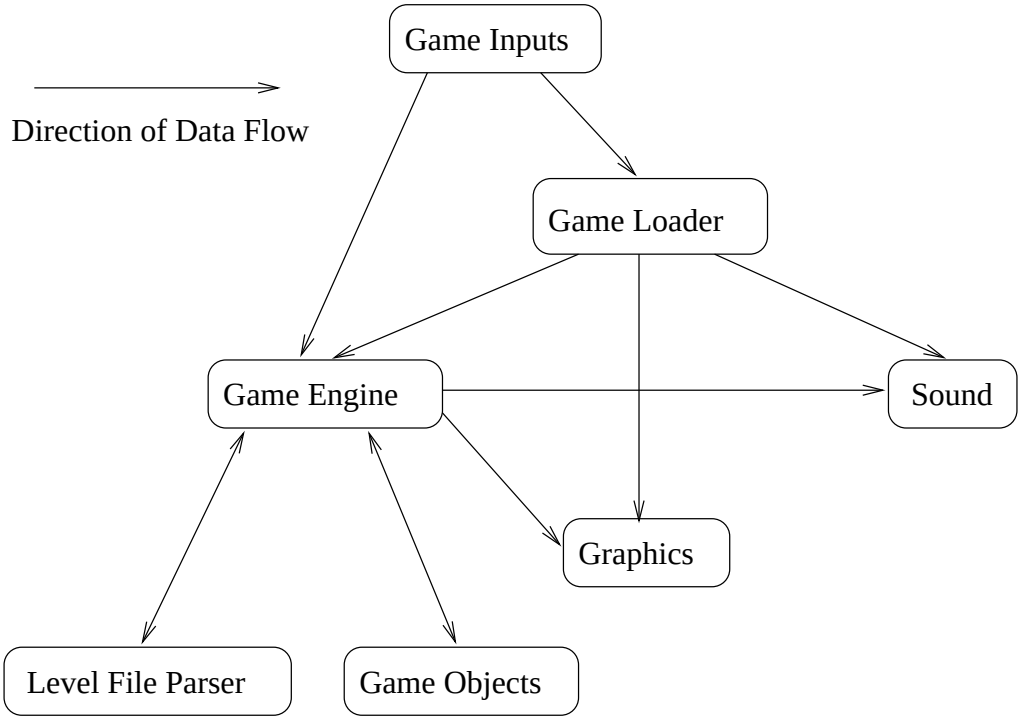
Catherine Hill: cjhill

1 Project Description

Trimuxs II is a 2D, side-scrolling, arcade-style shooter along the lines of Gradius games, R-Type games and Ikaruga. Playable by one or two players, Trimuxs II has players travelling through space and various worlds to destroy some great evil that has spread darkness across the galaxy. Of course, this evil has left its mark on these worlds and many emenies await to stop players from destroying their master. These enemies range from ground cannons, space fighters, and even the walls of a level itself. As players advance through levels and collect powerups to increase their offensive and defensive capabilities, they will constantly be dodging bullets shot by enemies. Should they survive a given level, they will end up face to face with a powerful boss who will undoubtably have some really powerful, large and possibly colorful attack or some other way of making at least half of the screen unsafe for the players. The game continues from level to level until the players succeed in destroying the darkness or are destroyed.

2 System Model

2.1 Diagram



2.2 Written Descriptions

- **Game Inputs:** The game inputs component is responible for obtaining a user's input and transmitting that data to the game engine or game loader in an understandable format.

- **Game Loader:** The game loader sontrols the whole game. It is responsible for causing the opening and ending scenes and obtaining all of the game configuration options from the data it gets from the game inputs. Once the configuration options are set, the game loader starts the game engire going and resumes control after the game is over. It displays the ending sequence and then starts again. This continues until the user exits.

- **Game Engine:** The game engine is responsible for running a single game. The configuration options are set from the game loader, and the game engine is then allowed free rule over the game. It gets a parsed level file from the level file parser and runs the level. It is responsible for making sure that it tells items when to move, fire, check for collisions etc... and also causes the sound component to play sounds and the graphics component to display all of the game objects currently on the screen. It also gets the user's input from the game input component to use to move player ships, fire, and pause.

- **Graphics:** The graphics component is responsible for displaying the images of the various game objects, backgrounds and other screens used during the course of the game.

- **Sound:** The sound component is responsible for playing the various music and sound effects throughout the game.

- **Game Objects:** The game objects are all of the various objects encountered in the levels:

  - Player ships - The ships controlled by the players
  - Enemies - All of the enemies in the game (includes bosses)
  - Walls etc... - The obstacles in the level including the upper/lower and/or right/left walls. These are things that do not actively shoot at the player, but normally cause the player to die should they collide.
  - Shots - The bullets fired by enemies and players that are currently on the screen.

  These objects are responsible for being able to:

  - Keep track of where they are
  - Move
  - Collide with other objects
  - Fire - only ships can do this

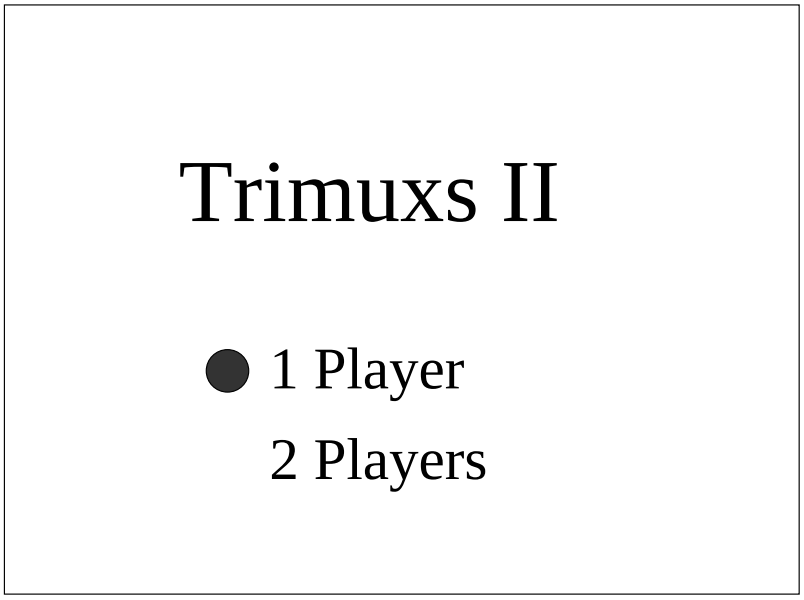
- Dissappear when they are off the screen and not coming back or when they are hit

- **Level File Parser:** The level file parser is responsible for parsing the game's level files into a format that the game engine can use to run the game. This probably includes creation of enemies, walls etc...

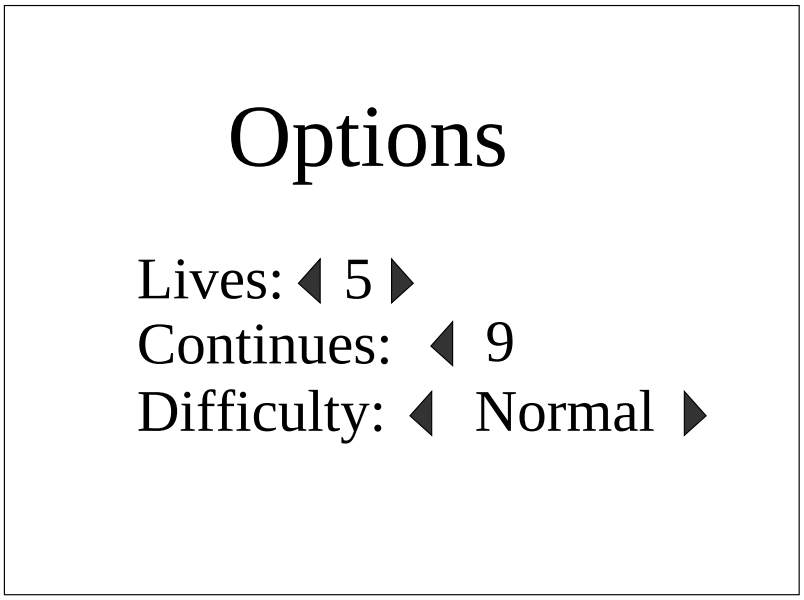
3 User Interface

3.1 Diagrams

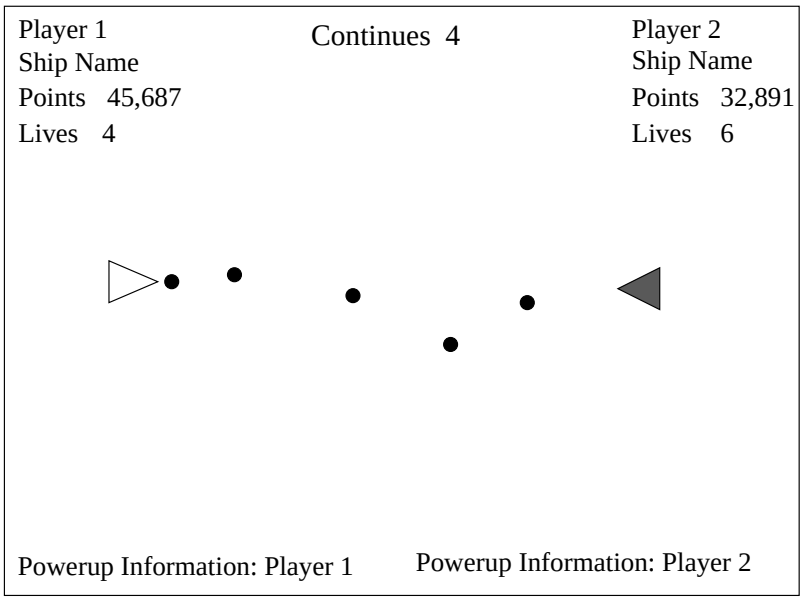
Opening Screen



Option Screen



Game Play Screen



3.2 Written Descriptions

- **Opening Screen:** This is where user(s) select how many players will be playing. Next they go to the option screen.
- **Option Screen:** This is where player(s) can select what options they will be using during the game.
  - **Lives:** The number of lives a player has each continue.
  - **Continues:** The number of continues the player(s) will get to share. This includes the initial set of lives each player uses.
  - **Difficulty:** The difficulty level of the game (easy, normal, hard).
- **Game Play Screen:** This is where the actual game play takes place. Players' numbers, ship names, point total and remaining lives are displayed in the upper right and left corners. The corrseponding player's powerup levels are displayed in the bottom corners of the screen. In the upper middle of the screen is a message indicating the number of continues the player(s) have remaining. The rest of the screen is devoted to the game (enemies, player ships, walls and bullets).

4 Non-functional Requirements

4.1 Performace

The game should run smoothly once an instance of the game has been started. Once the actual game begins, it should update at least 40 times per second in order to ensure smooth animation. As many bullets and enemy ships come onto the screen,

there should be minimal graphical slowdown (no slowdown is preferable, but there can be rare and circumstances when a small amount of slowdown occurs due to large amounts of objects being on the screen).

4.2 Testing

Each component of the system should be extensively tested before integration is started. Measures will need to be taken to make sure that there is at least a game engine prototype available for other components to use for parts of their testing as initial integration begins. Once the integration begins, probably by integrating the engine and another component, test suites will be needed to make sure that the functionality of each component is not reduced by the addition of another component. As more components are added, these test suites will need to be run again and again to make sure that no new component's addition destroys any current funcionality.

Once all components are integrated and the game is runable, it will need to be thoroughly play tested to make sure levels act and feel right. Enemies should not fly through walls and bosses should be on the screen at the right time. The overall effects of the game need to be synchronized as well so that the boss music is not playing during the beginning of a level. This will also be the time to try and break the game by trying as many wacky things as possible and making sure the game reacts accordingly (if you die and come back in a wall, you should die again after the usual few seconds of invulnerability and not stay in the wall safe from harm).

4.3 Reliability

The game should continue to run, once started, until the user(s) exits. The game should not spontaneously crash, and *if* there is an error it should ever reach the console. Instead, the current game in play should stop and the program should go back to the opening screen and let the user(s) play again.

4.4 Ease of Use

The resulting product should be very easy to use as its target audience is simply people who want to have some fun. The initial game configurations (number of lives etc...) should be presented in an interface where it is easy to select the different options desired for game play. Once the game begins, the controls should be straightforward and simple - 4 movement keys, a "pause" key and a "fire" key.

4.5 Portability

The only dependency this project has on other software is its dependency on the C++ SDL library. This library is available for both Windows and Linux, so ideally the project should easily port from Linux to Windows. [1]

[1] This requirement is made barring any unforseen serious Linux dependencies needed to make this project function on a Sunlab machine.

4.6 Documentation

The documentation for this project will be similar to an instruction guide for a computer game. There will be instructions for how to run the game from a console as well as instructions on how to play the game itself. The latter wil include detailed instructions for starting the game - selecting number of players, ships, and possibly lives, continues and difficulty mode - as well as detailed instructions for how to play - how to move, fire, and pause. Following this there will be a less detailed description of *some* of the enemies, weapons and levels in the game. There will also be the obligatory "plot" background summary near the beginning.

4.7 Dependencies on Other Systems

The dependencies for this projects lie in the external library, SDL. SDL would be used to: display graphics, play audio sounds, and deal with user input (joysticks or keyboard).

5 Updated Requirements

These are ranked on a scale of 1-10, 10 being of highest priority.

- Overall Game Options
  - Must have at least a rudimentary opening sequence and ending sequence (4)
  - Must be able to skip the opening sequence if there is one (9)
  - Difficulty settings, that control how fast enemies shoot and number of lives (5)
  - Continues, shared between players (7)
  - More advanced opening and ending sequences (2)
- Player Ships
  - Must be able to move left, right, up, and down (10)
  - Must be able to fire forwards (10)
  - Must die when any "enemy" object collides with it absent a shield (10)
  - Must be able to pick up powerups that appear (10)
  - Must respawn if dies with lives remaining (10)
  - 2+ unique ships (8)
- Point System
  - Each player will have a running total of points earned (9)

  - Each enemy is worth a certain number of points and those points are awarded to the player who destroys that enemy (9)
  - Each player's points reset when they continue, but not when they die (8)
  - After obtaining a set number of points, a player earns a free life (7)
  - A "High Scores" page is included when the game is over. Players whose scores exceed those on the list get to add their initials and the list is updated. This list persists between multiple instances of the game. (3)

- User Inputs
  - Must allow joypad support (USB) (6)

- Graphics
  - Basic graphics - above and beyond polygons using basic images (NES styles graphics) (10)
  - Advanced graphics, with explosions and flashy weaponry (5)
  - Background must be easily distinct from foreground (3)

- Engine
  - Must load levels and play them continuously (10)
  - Must have both single player and two player support (10)
  - Must enforce all game rules (10)

- Enemies
  - At least 3 different types per level (10)
  - Must move in different patterns (10)
  - Must shoot at player, or in a specific direction (10)
  - Enemies have larger weaponry (8)
  - Enemies have varying health (8)

- Bosses
  - Every level must have a boss at the end (10)
  - Game must have at least one "interesting" boss, that isn't just a fixture in the wall (7)
  - Bosses must fire various weapons in prescribed patterns, with possibly one weapon aiming at the player (10)
  - A level comprised of 3-5 bosses right one after another (3)

- Powerups

  - Must be at least 1 unique weapon per ship (10)
  - Basic weapon must exist (10)
  - Must be either collected or upgraded from powerups, system yet to be determined (10)
  - Some sort of shield, either frontal or aura, must exist (10)
  - Laser type weapon (8)
  - Missiles (5)
  - Multiple shields, selectable at startup (3)
  - "Options", or any small object hovering or moving around the player, shooting or protecting the player (7)

- Audio
  - Must have a sound effect for most game effects, including weapon fire, death (10)
  - Must have at least one set of music for the game (10)
  - Ideally will have different music for each level (9)
  - Ideally will have different music for each boss (8)

- Level Creation
  - Some method must exist for creating levels (10)
  - This method should be fairly easy to use, like an easy to use XML language (8)
  - A level editor, visual or not, that would allow anyone to make levels and input them (1)

6 Risky Parts

- Seeing as how this is a 2D-graphical game, someone with graphical experience will be neccessary.