Tech Section

Software

Ervisen will be put together using JAVA. This is because it's the language that most of the developers have experience in using. Development of content will be dependent on other programs. The sounds will be put together using Fruity Loops <http://www.fruityloops.com/> and Audacity <http://audacity.sourceforge.net/>. The files developed will then be played via a library within JAVA exactly which one will be determined by flexibility and simplicity.

The visuals are largely to be developed by Adobe Photoshop <http://www.adobe.com/products/photoshop/>, the GIMP <http://www.gimp.org/> and Blender <http://www.blender3d.org/> . For the various image files we will be using PNG files to allow the needed transparency.

Certain cinematic transitions will be generated in Blender and played through a JAVA Library chosen for the same reasons the that the sound player library will be chosen.

Engine

We will be programming our own engine in order to allow for it to have our own personal feel and function. However, this will limit the game in other ways, for we will have to solve each of the game structure problems as they are needed or come up.

The engine is centered on the tile representation of the world. Each tile represented by a png file and drawn in the right places to fit the isometric concept. The tiles are drawn in order to maintain proper visibility. The transparency of each image will help create the illusion of shadows and allow the inter-mingling of land tiles with people and object tiles. This is achieved by drawing the land tiles and then drawing everything on that tile. The tiles are drawn from left to right and then down the screen. A 2D array of integers gives direction to how the areas are to be drawn. The tiles are numbered the same way; they are scaled up from their base size when they are drawn. This gives the unified look of pixels being 3 times their size.

The map will also include areas of the map that are called triggers. These will be what allow the world to be dynamic. The triggers will be how the character changes or is changed by the world. Each trigger's location will be based off its relation to a temporary object, most likely a person. This list of temporary objects will be store separately from the unchanging world sprites. And might also incorporate some kind of scripted movement.

The control system within the game is keyboard based. Mouse interface will only be used to access any main window menu bar options, and loading and saving the game.

The keyboard commands will consist of the arrow keys and a few other keys to activate certain modes or options. One of the main functions is to activate what is being called “the power” which will extend the amount of drawn tiles and cover the screen in a slight color change.

There will be a multiple window system. The main window will be the one that depicts the character in the center of world and will have fixed dimensions (800x600). When a trigger is tripped it
may open a window that will then become the active window, preventing the character from moving until the window's function is complete. Examples of these windows are combat, dialog, and purchasing windows. The window's goal will be obvious. The player will always have the option of making a status window display and that window would not prevent the character from moving. The maximum number of windows displayed at any one time would be 3 (Main, Combat/Dialog/..., Status)

The combat window will enact a turn based combat. The character will be able to select from four options. Attack with the equipped weapon. Use a spell, or an item. Fleeing from combat will give a chance that the player can escape the combat, though the enemy will still be standing in the main window.

Between tiled-map locations the player will have the main window replaced by a map of the world. Using the arrow keys they will be able to cycle through the locations adjacent to their current location.

And choose one of them. Though other locations will be visible on the map, they won't be able to choose them. This will
help give the player a sense of distance.

The location files will be saved as a text file. With comma separated integer values defining the maps. Saving games will save the arrays of flags for the player and the array of triggers for each location.

The files will hold all the information to run a location, and the character will hold the information to adjust how each location is dealt with will be determined by the flags within the character.

**Class Breakdown**

Main Window: This class will act as the main interface for the user. It will always be available to the player while the game is running though other windows will take precedence and deny interaction while they are visible. The tiled world will be displayed during regular game play and the world map will be displayed while it is active. James will be the one programming
this class. For it requires a general understanding of the other classes.

ImageTile: This class contains an image or series of images. It contains their offsets and keeps track of what image needs to be drawn in order for idle animations and walk animations to work. This class will be the nitty gritty of the palette class. James will be working on this class.

Palette: When a tile needs to be drawn this class knows which Image tile to call on. It keeps a bank of ImageTiles arranged in an array to allow the buffered image to get the right image when given the ImageTile array location. It will organize the tiles according to lands tiles, object tiles and people tiles. Usually for a location to appear different the switching of the palette should be almost all the needs to be done. James will code this class.

MapTile: Stores the proper values to be fed to the palette in order to get ImageTiles to be layered properly by . James will be coding this class.

LocationMap: Provides a class for all of the tiled maps. Each map will be a different class to be used by the Main Window for displaying the tiled world. This class will store the 2D array of MapTiles, and tell them when to draw. James will be coding this class.

Illustration 5: MapTile UML Diagram

WorldInterface: This stores the world map and the city markers. Based off which cities the Character has visited it will be drawn in the main window when the the player is choosing
which city to goto next. James will be coding this class.

NPC : This class will store the information for the various monsters and people of Ervisen. Each instance will be one person or monster and will describe the pattern and types of triggers they have. The pattern will be based off their facing direction. William will be coding this class.

LinesofDialog : The responsibilities of this class will be to allow a NPC to have an ordered set of things to say. William will be coding this with the plot as a base.

Trigger : This will have a location relative to the NPC to which it belongs, and it's job is to tell instances of the Activatable interface to do their thing. The Trigger is set off before the character actually moves into the tile, thus preventing him from walking. It can have multiple Activatables in a list in the order of their Activation such as Dialog then Item or PlayCinema then GotoMap, this class will pass the character and the NPC to the proper Activatable. James is the main coder on this.

The Activatables : each have an activate method which will cause what they need to make to happen. Usually it will open another window, but sometimes it will do something else. The activate method within each will receive the character and the NPC that triggered it.. James is also the main coder on this.

CombatWindow : In this class the display of the combat will be handled. The combat rules will apply and be implemented. Most of the carnage will be behind the scene though there may be a short one or two line description of each move that happens. Robert will be responsible for this class.

DialogWindow : This is how the NPC's present most of the story. Interaction will make the world seem larger and more complex. Witty jokes and tales of sorrow abound. It is possible to get items here, and the character's flags will be very important here. Robert has this challenge.

PurchaseWindow : Trading money for items. The economy will rely on this method of communicating what an item is and how much it costs. Robert's responsibility

ItemWindow : Sometimes there is something just laying around, so, in a way this is a purchasing window without money. Robert will be coding this one.

SoundPlayer : Sets up and handles the threads involved in making Ervisen multimedia during game play. Since Ian has put so much into the sound of the game he will be in charge of getting it through the player.

PlayCinema: is kind of special as an Activatable in that it shows an AVI in a window which should close once the video is done. Ian will be handling this.

GotoMap : Switches the location to a new one. It's used to travel from the town to a store or another building, or even another town without using the World map.
GotoWorld: Will be why the map switches from a town to the over map. This will be mainly handled by James.

Flagger: This will maintain the array of flags, and will be what many of the activatables will be interested in getting information from and to. This will be William's responsibility.

Character: This is the hero, with items, attributes and a set of flags managed by a Flagger class. Any changes in this class could have some serious ramifications for the game. William will be tackling this one.

Item: This class gives the Character something to buy or carry. This class will handle any items that are solely used as flags or jokes. Robert will be handling this class and working closely with Will.

Weapon, Armor, Misc: various types of items that have different properties. Weapons affecting attacks. Armor affecting defense. Misc items can be used only once. Robert will be coding this class.

StatusWindow: This window is how the player sees the Character class and manages the various items. This will be important to and done by William.

ScriptedMovement: is something that might be added if the problem somehow proves simpler than it initially seems. William would be working on this, with James.

**Use Case Diagram**

![Use Case Diagram](image-url)
Illustration 7: Movement Diagram