

System Specification

Introduction

Goals and Objectives

GameForge is a graphical tool used to aid programmers in the design and creation of video games. A game designer with little or no experience with Microsoft DirectX and/or Visual C++ programming can use GameForge to construct his or her own 2D-arcade games. GameForge also assists experienced programmers by generating the Microsoft DirectX code and Microsoft Windows9x overhead necessary for basic game construction, allowing them to concentrate on more detailed game design issues and implementation. The idea is to limit the amount of actual code written by the game designer, while providing an interface that is easy to use yet complete enough to remain functional.

As a short-term goal, GameForge is intended to assist students in completing their final game design assignments in CIS 587. As a long-term goal, PA Software hopes GameForge will be among the first in a growing selection of 'do-everything' game builders.

System Statement of Scope

GameForge is a graphical tool used to aid in the design and creation of video games. A user with limited Microsoft DirectX and/or Visual C++ programming knowledge will be able to construct a basic 2D-arcade game. The idea is to limit the amount of actual code written by the user. It will also assist experienced programmers in generating the Microsoft DirectX and Microsoft Windows9x overhead necessary for basic game construction, allowing them to concentrate on more detailed game design issues and implementation.

The software will consist of a number of inputs, graphically assisting the user in creating on-screen objects including the following:

- User Created Objects (player character, creatures, static objects)
 - Bitmaps (with animation)
 - Collision Detection Areas
 - Movement Routines
 - Additional Object Attributes
- Backgrounds
- Input Device Setup
- Sound Events

The software will also consist of a number of graphical processing functionalities including the following:

- Defining/Editing Objects (including characteristics)
- Object Positioning
- Opening/Closing/Saving Game Project Files
- Exporting Game Projects to compilable C++ Files

Outputs include:

- User Created Sprite Objects
- Bitmaps
- Flat text data files that are readable by the Microsoft VC++ engine
- Game Project Files
- Visual C++ header files containing necessary system initializations
- Database Files

System Context

GameForge is being marketed as a CASE tool, to allow software developers to 'build' rather than code their game. It is not necessary for developers to have prior knowledge with DirectX or Visual C++, as long as they have a good art team and high production values. GameForge will be commercially distributed via the GameForge website (for information regarding the URL, see the Appendix.)

GameForge will be available free for educational use. It will be distributed for use in CIS 587, Computer Game Design and Implementation, at the University of Michigan-Dearborn.

Major Constraints

Performance/Behavior Issues

GameForge is designed to be compatible with the Microsoft Windows 9x operating system. Microsoft Windows NT 4.0 and earlier versions will not be supported (Windows NT only supports Microsoft DirectX up to version 3.0. DirectInput had not been implemented at this time, making this version of DirectX very limited.) Microsoft Windows 2000 should also be compatible.

GameForge also requires Microsoft DirectX 7.0 or above. Users may also want to obtain the DirectX 7.0 SDK if they plan on expanding the GameForge library files beyond their original scope.

GameForge also requires the Microsoft Visual C++ 6.0 compiler. GameForge's VC++ code may be compilable using Borland or some other VC++ compiler, but functionality is not guaranteed.

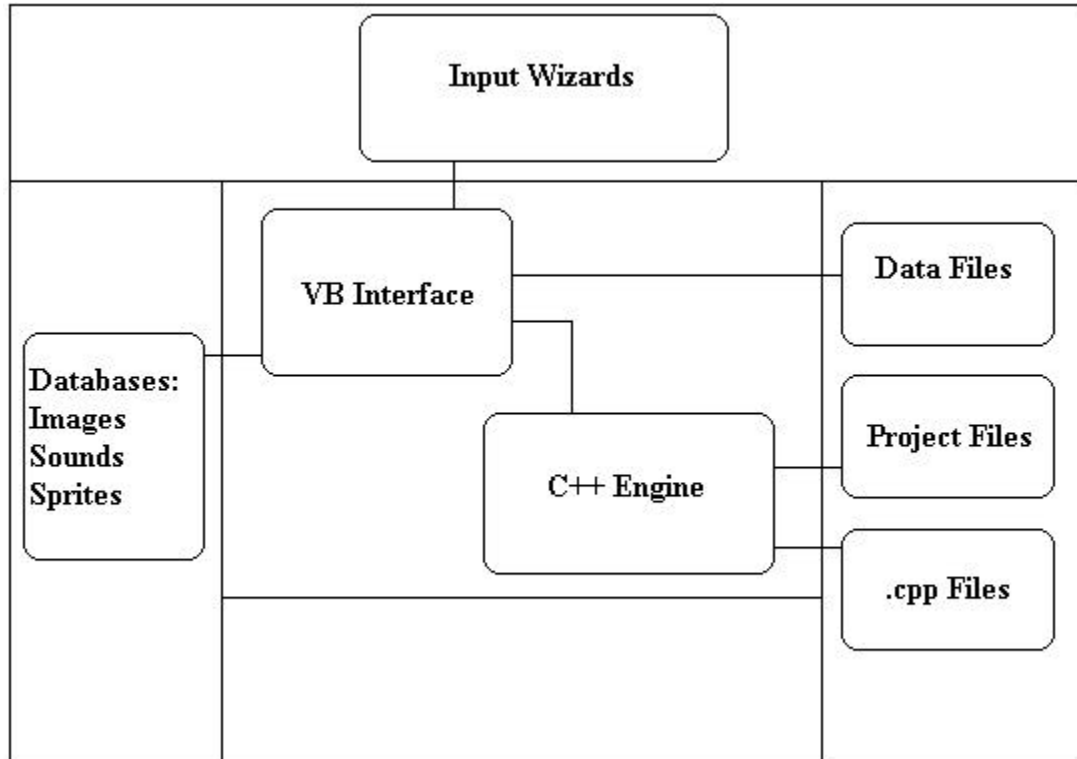
Management and Technical Constraints

GameForge has a drop-dead delivery date of 04/17/00.

Functional and Data Description

System Architecture

Architecture Model



Subsystem Overview

The following page includes brief descriptions of each subsystem illustrated on the Architecture model, as well as the nature of their specific interfaces.

User Interface Processing

- Input wizards –
 - There are a number of wizards provided to guide the novice user through the necessary steps for game development.
 - New World Wizard –

This wizard allows the user to begin creating a new world. General settings (like screen resolution and transparent color) are set here.

- New Level Wizard –
This wizard allows the user to create a new level within their game. Level specific attributes (like background color and level size) are set here.
- New Sprite Wizard –
This wizard assists the user in the creation of a new sprite. The user can select attributes and movement options.
- Sprite Wizard (Animation Wizard) –
The user is assisted in the animation process. They can create an animation cycle frame by frame, and test the result.
- Sprite Wizard (Collision Area Wizard) –
The user sets the collision area. It is displayed over the sprite.
- Level Editor –
The main interface displays the level/game the user is designing. A tree-view of all objects is also represented. All other interface options are accessed from this screen.
- Help/Tutorial Files –
These files contain FAQ's, Tutorial, descriptions of objects and VC++ code, and a search engine to find the information.

Input Processing

- Databases – GameForge utilizes a Microsoft Access database to store sound libraries and image libraries, as well as pre-designed sprites. The database is accessed by the user interface.

Process and Control Functions

- VB interface – The interface is the subsystem the user interacts with. It creates a project space for all project files to be stored in. It gathers all necessary data from the user, as well as interacting with the access databases. The interface then generates data files containing all specifications of all the

sprites, as well as input device information and sound information. All necessary files such as .wav files and .bmp files are moved to the project directory. This subsystem contains the screen representing the game and a list of all sprites and their attributes.

- C++ engine – This subsystem is the meat of the system. The engine creates a .cpp file for the game. The file contains references to the data files generated by the user interface and references to DirectX code contained in header files.

Output Processing

- Flat Text Data files – a file containing information specified by the user that is read by the C++ code. The files are generated by the user interface. The user's game can be tweaked by editing this file rather than rewriting code. These files include:
 - World Information
 - Sprite Information
 - Player Information
- Program Files – Files are stored with a unique extension to be used exclusively by GameForge. This file is actually a Microsoft Access database, and is generated as temporary storage during game creation. They are generated by the user interface.
- .cpp Files – Finished projects can be saved as a .cpp file that can be compiled with Microsoft's Visual C++ compiler to create an executable file for the game. This file is generated by the C++ engine.

Maintenance and Self-Test

- No maintenance subsystems are planned for this version of GameForge. However, any patches that are released after GameForge will be available from the official GameForge website.

Data Description

Major Data Objects

Sprites:

Sprites consist of the following attributes:

Name (Primary Key) – Name of the sprite.
Image – Name of the Image file displayed representing the sprite.
Width – The width of the sprite in pixels.
Height – The height of the sprite in pixels.
DestinationX – The X coordinate for the destination of the placement of the sprite.
DestinationY – The Y coordinate for the destination of the placement of the sprite.
Framerate – The framerate of the sprite.
NumOfDir – The number of directions the sprite has.
NumOfFrames – The number of frames per direction.
Solid – Whether or not the sprite is a solid object.
KillsPlayer – Whether or not the sprite kills the player.
PlayerCanKill – Whether or not the player can kill this sprite.
OtherCanKill – Whether or not other sprites (other than the player) can kill this sprite.
Obtainable – Whether or not the sprite can be picked up.
Visible – Whether or not the sprite is visible on the screen.
AffectsScore – Whether or not the sprite has an effect on the score.
SoundAttached - Index of the sound is attached to this sprite.
ReactsToGravity – Whether or not the sprite is affected by gravity.
ReactsToFriction – Whether or not the sprite is affected by Friction.
ReactionToPlayer - Index determining how the sprite reacts to the player's position.
Bounces – Whether or not the sprite changes direction when contacting another sprite.
Random – Whether or not the sprite has random movement.

Surfaces:

Surface consist of the following attributes:

Alias (Primary Key) – The filename of the image.
Path – The directory that the image is found in.
Height – The height of the image in pixels.
Width – The width of the image in pixels.

Messages:

Messages consist of the following attributes:

String (Primary Key) – The actual text to be displayed.
DestX - The X coordinate for the destination of where the string will be placed.

DestY – The Y coordinate for the destination of where the string will be placed.

ForeRGB – The RGB value of the foreground color of the text.

BackRGB – The RGB value of the background color of the text.

Transparent – The background of the text can be made transparent.

Visible – Whether or not the text is visible on the screen.

Sounds:

Sounds consist of the following attributes:

Alias (Primary Key) – The filename of the sound.

Path – The directory that the sound is found in.

Buffer – The buffer to load the sound onto.

Notifications – Breaks up sound into pieces for streaming.

State – Determines play state (playing, not playing, looping).

Levels:

Levels consist of the following attributes:

Name (Primary Key) – The name of the level.

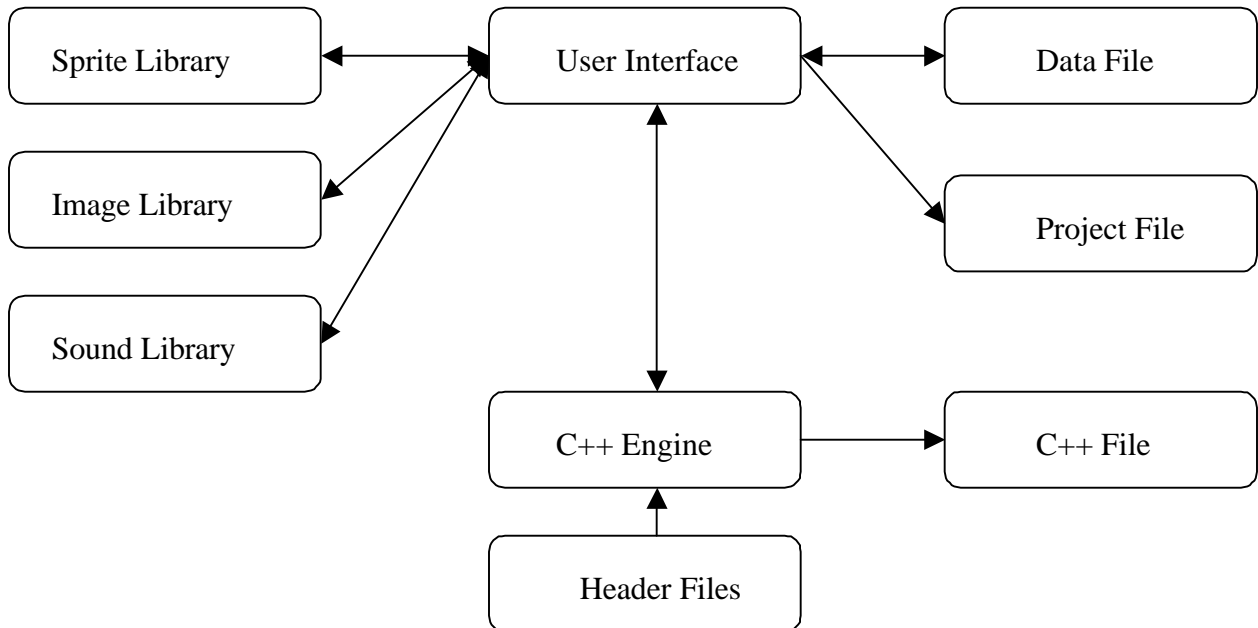
Width Overall width of the level (in pixels).

Height Overall height of the level (in pixels).

GoalSprite Index of the goal sprite.

GoalScore Value of the goal score.

Relationships



Interface Description

External Machine Interfaces

None identified.

External System Interfaces

GameForge requires a number of additional system components to operate. These include the following:

- Microsoft Visual Basic runtimes - necessary to use the user interface
- Microsoft DirectX 7 libraries - these include information needed to compile and run a created game.
- Microsoft Visual C++ - required to compile .cpp files to create an executable file for the game.
- Any text editor – used to edit data file if necessary

User Interface

The interface will have on the left side of the screen, a treeview control, which displays elements in a directory tree structure. Placed in this treeview will be categories with which the sprites that user has input will

be categorized. The user will be able to click on these categories and see the expanded list of the sprites that are under that particular category. In addition the user will be able to click on a particular sprite and bring up all of that sprite's properties. When the sprite wizard is up, the right side of the screen will be the sprite's image, and an area where that image can be placed on the screen, onto any of the particular backgrounds that the user has chosen. There will be a standard menu bar that is present on nearly every Microsoft Windows application, along with a toolbar for quicker access to the commands embedded under the various menu options.

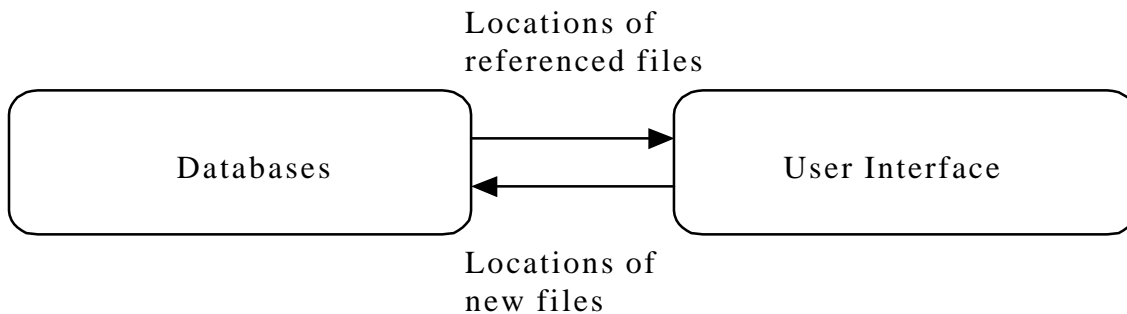
Subsystem Description

Description for Access Databases

Subsystem Scope

There are three databases used by GameForge, one for sprites, one for images, and one for sounds. Each database contains the locations of files, either data files describing the sprite, bitmap files for images, or sound files in .wav format.

Subsystem Flow Diagram



Access Database Components

Sprite Database Description (Processing Narrative)

The sprite database contains the locations of data files. These data files contain specifications for sprites, either pre-defined, or user-defined. This allows users to use existing sprites in their games, or to save the sprites they create.

Sprite Database Interface Description

The sprite database is referenced and written to be the user interface. The interface is written in Microsoft Visual Basic, which allows for easy interaction with Microsoft Access databases.

Image Database Description (Processing Narrative)

The image database contains the locations of bitmap image files. These files are the part of the sprite that is seen on screen. They can be files supplied by GameForge, or can be created by the user.

Image Database Interface Description

The image database is referenced and written to be the user interface. The interface is written in Microsoft Visual Basic, which allows for easy interaction with Microsoft Access databases.

Sound Database Description (Processing Narrative)

The sound database contains the locations of sound files in .wav format. These files can be supplied by the system, or created by the user. The sounds are tied to events in the game.

Sound Database Interface Description

The sound database is referenced and written to be the user interface. The interface is written in Microsoft Visual Basic, which allows for easy interaction with Microsoft Access databases.

Performance Issues

The C++ engine must access the database. Any problems encountered in this interface must be minimized.

Design Constraints

The databases contain only the locations of the files. Problems could occur if the files are moved. Special note should be included in the manual instructing the user to not delete files from the libraries.

Allocation for Access Databases

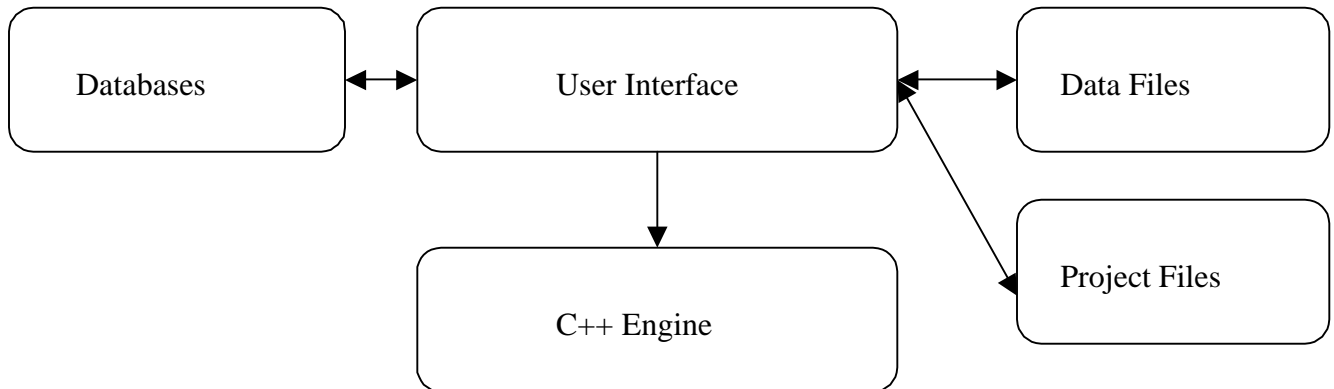
The databases will be created using Microsoft Access, and referenced by Microsoft Visual Basic.

Description for User Interface

Subsystem Scope

The user interface must present the user with an easy to use way to implement game design. It is a GUI that shows the user where things are, and what they do in the game. It gives the novice user a better understanding of what happens in the game. Wizards are supplied to guide the user through all necessary steps in game construction. Game design is not a primary function of GameForge, preliminary design should be completed before construction.

Subsystem Flow Diagram



User Interface Components

Sprite wizard Description (Processing Narrative)

The sprite wizard is a tool that guides the user through the creation of a sprite. It prompts the user for a number of inputs including the following:

- Bitmap image for the sprite
- Initial location of the sprite
- Sprite movement
- If the sprite is solid or not
- Visibility of the sprite
- Many other options

The wizard then creates a data file containing the information supplied by the user. Any necessary files are moved to the project space.

Sprite wizard Interface Description

The sprite wizard is a part of the user interface; it interacts directly with the databases to gather necessary information. It then moves the information to the game data file in the project space.

Project wizard Description (Processing Narrative)

The project wizard guides the user through the creation of the primary surface, sprite placement, game logic, and input. It prompts the user for information on these subjects, which is written to a data file in the project space.

Project wizard Interface Description

The project wizard is a part of the user interface; it interacts directly with the databases to gather necessary information. It then moves the information to the game data file in the project space.

Performance Issues

The primary performance requirement is a screen refresh rate of 30 frames per second.

Design Constraints

No special design constraints impact the user interface subsystem.

Allocation for User Interface

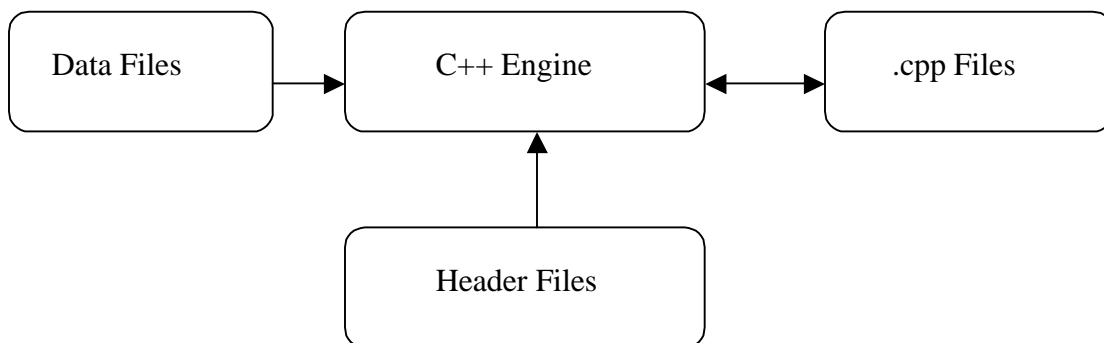
The interface will be implemented in Visual Basic. This will ensure proper interaction with the databases. It will also utilize DirectX to help translate the user's game to executable code.

Description for C++ Engine

Subsystem Scope

The C++ engine is the meat of the system. It is designed to be as modular as possible. A majority of the DirectX code is contained in header files to limit what has to be written by the engine. It gathers information from the data files, and uses it to create the balance of the code needed for the user's game.

Subsystem Flow Diagram



C++ Engine Components

Window Creation Component Description

This component is responsible for the overhead involved in the creation of a window for the user's game to operate in. It allows the size to be specified, as well as a title for the window.

Window Creation Component Interface Description

The necessary code for window creation will be contained in a header file. Any additional information will be extracted from the data file created by the user interface.

DirectX Component Description

This component handles all DirectX functions used in GameForge. These include DirectDraw, DirectInput, and DirectSound. These functions are responsible for the largest part of the game files. All on screen movement, all images, all sounds, are a result of the DirectX components.

DirectX Component Interface Description

The necessary code for the DirectX components will be contained in a header file. Any additional information needed will be extracted from the data file created by the user interface.

Data File Component Description

This component reads the data file created by the user interface, and uses the information to adapt the C++ code to the user's game.

Data File Component Interface Description

The component interacts with the data file created by the user interface. The information in the document is used to create a main file for the game.

Game Logic Component Description

The game logic component keeps track of what every sprite and sound does, and when. it uses the data file to dictate the process order for all objects in the game. The logic can be modified directly in the .cpp file if necessary.

Game Logic Component Interface Description

The game logic component is created in the same way as the other C++ components. The necessary header files are accessed. Any holes in the code are filled with information gathered from the game data file.

Performance Issues

The refresh rate of 30 frames per second must be maintained to have a playable game.

Design Constraints

The code created by the C++ Engine must be compatible with Microsoft's Visual C++ compiler, and DirectX 7.

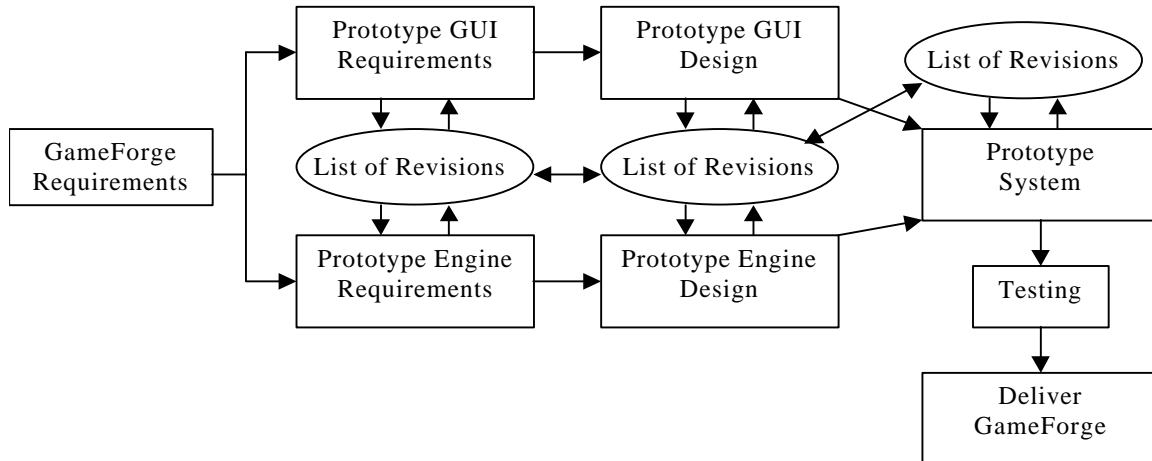
Allocation for C++ Engine

The engine will be created using Microsoft Visual C++.

System Modeling and Simulation Results

Description of System Modeling Approach

System modeling takes place during early team meetings. No special tools will be used. The model will be developed on paper.



Simulation Results

No simulations were used.

Special Performance Issues

Refresh rate of 30 frames per second.

Prototyping Requirements

The interface prototype will be created using Microsoft Visual Basic. The C++ portion will be developed in modules that will be individually tested.

Project Issues

Projected Development Costs

Function Point estimations:

Interface: 243.04

Engine: 303.45

Total FP: **546.49**

Reference FP calculations:

Demon Tree FP: 121.03

Demon Tree Person Months: 2.5

Estimated Person Months:

Avg. FP per Person Month: 48.412

GameForge est. Person Months: 11.288

Cost in Person months:

Industry average cost per Person Month: \$8,000.00

GameForge est. total cost (w/o equipment): **\$90,306.54**

Project Schedule

- Engine construction / Interface construction / Database construction
- System integration
- Alpha testing / customer evaluation
- Library creation
- Help/tutorial construction / manual
- Beta testing / customer evaluation
- Documentation revision
- Delivery

Note: dates have yet to be established.

Appendices

Product Strategies

GameForge may be distributed freely or as low-cost shareware. This decision will be made nearer to the project's completion. If GameForge is distributed as low-cost shareware, Professor Bruce Maxim will receive a free educational copy for use with CIS 587 at University of Michigan – Dearborn. The software will also be posted on PA Software's web site. There will be no additional commercial distribution of the system.

Supplementary Information

None at this time.