SOFTWARE REQUIREMENTS SPECIFICATION

Super Mountaineer Brothers

2/14/2010

West Virginia University

CS430

Anthony Palma, Jason Hooks, Adam Trainer, Nick Fleming

Contents

[1. Introduction 6](#_Toc253906455)

[1.1 Goals and objectives 6](#_Toc253906456)

[1.2 Statement of scope 6](#_Toc253906457)

[1.3 Software context 6](#_Toc253906458)

[1.4 Major constraints 7](#_Toc253906459)

[2. Usage scenario 7](#_Toc253906460)

[2.1 User profiles 7](#_Toc253906461)

[2.2 Use-cases 9](#_Toc253906462)

[2.3 Special usage considerations 9](#_Toc253906463)

[3. Data Model and Description 9](#_Toc253906464)

[3.1 Data Description 9](#_Toc253906465)

[3.1.1 Data objects 10](#_Toc253906466)

[3.1.2 Relationships 10](#_Toc253906467)

[3.1.3 Complete data model 10](#_Toc253906468)

[3.1.4 Data dictionary 11](#_Toc253906469)

[4. Functional Model and Description 12](#_Toc253906470)

[4.1.1 Description for Function “initSounds” 13](#_Toc253906471)

[4.1.1.1 Processing narrative (PSPEC) for function 13](#_Toc253906472)

[4.1.1.2 Function interface description 13](#_Toc253906473)

[4.1.1.3 Performance Issues 13](#_Toc253906474)

[4.1.1.4 Design Constraints 13](#_Toc253906475)

[4.1.2 Description for Function “drawScene” 14](#_Toc253906476)

[4.1.2.1 Processing narrative (PSPEC) for function 14](#_Toc253906477)

[4.1.2.2 Function interface description 14](#_Toc253906478)

[4.1.2.3 Performance Issues 14](#_Toc253906479)

[4.1.2.4 Design Constraints 14](#_Toc253906480)

[4.1.3 Description for Function “drawPlayer” 15](#_Toc253906481)

[4.1.3.1 Processing narrative (PSPEC) for function n 15](#_Toc253906482)

[4.1.3.2 Function interface description 15](#_Toc253906483)

[4.1.3.3 Performance Issues 15](#_Toc253906484)

[4.1.3.4 Design Constraints 15](#_Toc253906485)

[4.1.4 Description for Function “AI” 16](#_Toc253906486)

[4.1.4.1 Processing narrative (PSPEC) for function 16](#_Toc253906487)

[4.1.4.2 Function interface description 16](#_Toc253906488)

[4.1.4.3 Performance Issues 16](#_Toc253906489)

[4.1.4.4 Design Constraints 16](#_Toc253906490)

[4.1.5 Description for Function “boundaryTests” 17](#_Toc253906491)

[4.1.5.1 Processing narrative (PSPEC) for function 17](#_Toc253906492)

[4.1.5.2 Function interface description 17](#_Toc253906493)

[4.1.5.3 Performance Issues 17](#_Toc253906494)

[4.1.5.4 Design Constraints 17](#_Toc253906495)

[4.1.6 Description for Function “printToScreen” 18](#_Toc253906496)

[4.1.6.1 Processing narrative (PSPEC) for function 18](#_Toc253906497)

[4.1.6.2 Function interface description 18](#_Toc253906498)

[4.1.6.3 Performance Issues 18](#_Toc253906499)

[4.1.6.4 Design Constraints 18](#_Toc253906500)

[4.1.7 Description for Function “smoothMoves” 19](#_Toc253906501)

[4.1.7.1 Processing narrative (PSPEC) for function 19](#_Toc253906502)

[4.1.7.2 Function interface description 19](#_Toc253906503)

[4.1.7.3 Performance Issues 19](#_Toc253906504)

[4.1.7.4 Design Constraints 19](#_Toc253906505)

[4.1.8 Description for Function “jump” 20](#_Toc253906506)

[4.1.8.1 Processing narrative (PSPEC) for function 20](#_Toc253906507)

[4.1.8.2 Function interface description 20](#_Toc253906508)

[4.1.8.3 Performance Issues 20](#_Toc253906509)

[4.1.8.4 Design Constraints 20](#_Toc253906510)

[4.1.9 Description for Function “display” 21](#_Toc253906511)

[4.1.9.1 Processing narrative (PSPEC) for function 21](#_Toc253906512)

[4.1.9.2 Function interface description 21](#_Toc253906513)

[4.1.9.3 Performance Issues 21](#_Toc253906514)

[4.1.9.4 Design Constraints 21](#_Toc253906515)

[4.1.10 Description for Function “idle” 22](#_Toc253906516)

[4.1.10.1 Processing narrative (PSPEC) for function 22](#_Toc253906517)

[4.1.10.2 Function interface description 22](#_Toc253906518)

[4.1.10.3 Performance Issues 22](#_Toc253906519)

[4.1.10.4 Design Constraints 22](#_Toc253906520)

[4.1.11 Description for Function “drawText” 23](#_Toc253906521)

[4.1.11.1 Processing narrative (PSPEC) for function 23](#_Toc253906522)

[4.1.11.2 Function interface description 23](#_Toc253906523)

[4.1.11.3 Performance Issues 23](#_Toc253906524)

[4.1.11.4 Design Constraints 23](#_Toc253906525)

[4.1.12 Description for Function “special\_down” 24](#_Toc253906526)

[4.1.12.1 Processing narrative (PSPEC) for function 24](#_Toc253906527)

[4.1.12.2 Function interface description 24](#_Toc253906528)

[4.1.12.3 Performance Issues 24](#_Toc253906529)

[4.1.12.4 Design Constraints 24](#_Toc253906530)

[4.1.13 Description for Function “special\_up” 25](#_Toc253906531)

[4.1.13.1 Processing narrative (PSPEC) for function 25](#_Toc253906532)

[4.1.13.2 Function interface description 25](#_Toc253906533)

[4.1.14.3 Performance Issues 25](#_Toc253906534)

[4.1.14.4 Design Constraints 25](#_Toc253906535)

[4.1.17 Description for Function “keyboardUp” 26](#_Toc253906536)

[4.1.17.1 Processing narrative (PSPEC) for function 26](#_Toc253906537)

[4.1.17.2 Function interface description 26](#_Toc253906538)

[4.1.17.3 Performance Issues 26](#_Toc253906539)

[4.1.17.4 Design Constraints 26](#_Toc253906540)

[4.1.18 Description for Function “keyboard” 27](#_Toc253906541)

[4.1.18.1 Processing narrative (PSPEC) for function 27](#_Toc253906542)

[4.1.18.2 Function interface description 27](#_Toc253906543)

[4.1.18.3 Performance Issues 27](#_Toc253906544)

[4.1.18.4 Design Constraints 27](#_Toc253906545)

[4.1.19 Description for Function “CreateGlutWindow” 28](#_Toc253906546)

[4.1.19.1 Processing narrative (PSPEC) for function 28](#_Toc253906547)

[4.1.19.2 Function interface description 28](#_Toc253906548)

[4.1.19.3 Performance Issues 28](#_Toc253906549)

[4.1.19.4 Design Constraints 28](#_Toc253906550)

[4.1.20 Description for Function “CreateGlutCallbacks” 29](#_Toc253906551)

[4.1.20.1 Processing narrative (PSPEC) for function 29](#_Toc253906552)

[4.1.20.2 Function interface description 29](#_Toc253906553)

[4.1.20.3 Performance Issues 29](#_Toc253906554)

[4.1.20.4 Design Constraints 29](#_Toc253906555)

[4.1.21 Description for Function “InitOpenGL” 30](#_Toc253906556)

[4.1.21.1 Processing narrative (PSPEC) for function 30](#_Toc253906557)

[4.1.21.2 Function interface description 30](#_Toc253906558)

[4.1.21.3 Performance Issues 30](#_Toc253906559)

[4.1.21.4 Design Constraints 30](#_Toc253906560)

[4.1.22 Description for Function “main” 31](#_Toc253906561)

[4.1.22.1 Processing narrative (PSPEC) for function 31](#_Toc253906562)

[4.1.22.2 Function interface description 31](#_Toc253906563)

[4.1.22.3 Performance Issues 31](#_Toc253906564)

[4.1.22.4 Design Constraints 31](#_Toc253906565)

[4.2 Software Interface Description 32](#_Toc253906566)

[4.2.1 External machine interfaces 32](#_Toc253906567)

[4.2.2 External system interfaces 32](#_Toc253906568)

[4.2.3 Human interface 32](#_Toc253906569)

[4.3 Control flow description 32](#_Toc253906570)

[5. Behavioral Model and Description 32](#_Toc253906571)

[5.1 Description for software behavior 32](#_Toc253906572)

[5.1.1 Events 32](#_Toc253906573)

[5.1.2 States 35](#_Toc253906574)

[5.2 State Transition Diagrams 36](#_Toc253906575)

[5.3 Control specification (CSPEC) 38](#_Toc253906576)

[6. Restrictions, Limitations, and Constraints 39](#_Toc253906577)

[7. Validation Criteria 39](#_Toc253906578)

[7.1 Classes of tests and Responses 39](#_Toc253906579)

[7.1.1 Incorrect keyboard input 39](#_Toc253906580)

[7.2 Performance bounds 39](#_Toc253906581)

[8. Appendices 39](#_Toc253906582)

[8.1 Product Strategies 39](#_Toc253906583)

[Figure 1 Profile Seperation 8](#_Toc253906584)

[Figure 2 Use Case Diagram 9](#_Toc253906585)

[Figure 3 Data Persistence Relationship 10](#_Toc253906586)

[Figure 4 ERD 10](#_Toc253906587)

[Figure 5System State Transition Diagram 37](#_Toc253906588)

[Figure 6 Game in Session State Transition Diagram 38](#_Toc253906589)

# 1. Introduction

This section provides an overview of the entire requirement document. This document describes all data, functional and behavioral requirements for software.

## 1.1 Goals and objectives

The goals and objectives of developing Super Mountaineer Brothers are two-fold. First and foremost, recreating the game-play associated with the Super Mario Bros. platform video game is paramount to the success of this project. In mirroring the game-play of Super Mario Bros., the player familiar with this franchise will have a significantly shorter learning curve, while the various aspects of the Super Mario Bros., such as the red and yellow mushrooms, the Koopa Troopa, and Mario’s primary attack of jumping on enemies, will provide a basis for contrasting with the Super Mountaineer Brothers. With this in mind, implementing Mountaineer-specific enemies, weapons, and levels is equally important. From throwing beer cans, encountering Pitt Panthers, and riding the PRT, these differences with Super Mario Bros. will ensure an enjoyable gaming experience for the Mountaineer fan as well as the novelty of the game.

## 1.2 Statement of scope

The Super Mountaineer Brothers will offer game-play similar to that of the Super Mario Bros. The user’s keyboard input control the Mountaineer’s ability to move left and right, jump and duck, collect and throw objects. Regarding game functionality, users will be able to start a new game, pause a game, resume a game, and quit a game. Lastly, the computer output will consist of the user’s points, the user’s lives, and the generation of levels, enemies, and obstacles.

## 1.3 Software context

Since the Super Mountaineer Brothers will mirror many of the aspects of the Super Mario Bros., the distribution of the game will be extremely limited so as to avoid legal action from Nintendo. Super Mountaineer Brothers will be developed simply for the enjoyment of the designers and other Mountaineers.

## 1.4 Major constraints

As described above, the major constraints associated with the Super Mountaineer Brothers are the possibility of legal action from Nintendo. To avoid this, all aspects of its development, testing, and game-play will be restricted to the developers.

# 2. Usage scenario

This section provides a usage scenario for the software. It organized information collected during requirements elicitation into use-cases.

## 2.1 User profiles

Our system has two user profiles, there are users and developers. A user is anyone playing the game in a release build. A developer is a user that can run the game in debug mode in addition to release mode.



Figure Profile Seperation

## 2.2 Use-cases



Figure 2 Use Case Diagram

## 2.3 Special usage considerations

This software will run in a windows environment using OpenGL and SDL. To run in debug mode you must have access to the source code.

# 3. Data Model and Description

This section describes information domain for the software

## 3.1 Data Description

Data objects that will be managed/manipulated by the software are described in this section.

### 3.1.1 Data objects

Our project will store data as game saves. The game saves will likely be in the form of a simple text file. The game save will contain various player attributes: the brother, lives, level, coins, etc.

Levels will be stored in resource files. Each level will contain a texture background image and a general layout of blocks, enemies, pipes, prt cars, etc. Levels will be made in a separate level editor program. Blocks, enemies, and pipes will be common among multiple levels so the actual textures and graphically related data for these elements will not be stored in the level files.

### 3.1.2 Relationships



Figure Data Persistence Relationship

### 3.1.3 Complete data model

The following is an ERD describing our system.



Figure ERD

### 3.1.4 Data dictionary

The data for Super Mountaineer Brothers will be kept in a local data store on each user’s machine. It will contain saved game data.

# 4. Functional Model and Description

Major Software Components used:

C++ programming language – main component of development

OpenGL – basic graphical rendering libraries

glut – advanced graphical rendering libraries

SDL – audio input and output libraries that work well with OpenGL/glut

SMB.cpp

## 4.1.1 Description for Function “initSounds”

initSounds – initialize SDL sound files

### 4.1.1.1 Processing narrative (PSPEC) for function

Function will initialize the SDL sound functions by importing sound files from the source folder

### 4.1.1.2 Function interface description

Function will take .wav files as input and allow for sound output

### 4.1.1.3 Performance Issues

There is a slight delay in sound output. We are not sure if this is an issue with SDL or with the use of chunk sounds.

### 4.1.1.4 Design Constraints

Only a finite number of sounds can be implemented (limited to the number of MIX chunks allowed by the system

## 4.1.2 Description for Function “drawScene”

drawScene – draw the scene

### 4.1.2.1 Processing narrative (PSPEC) for function

Function will render every stationary object in the scene, which includes the background, ground, platforms, and other various objects.

### 4.1.2.2 Function interface description

This function takes no parameters and outputs graphical renderings.

### 4.1.2.3 Performance Issues

We must determine whether we want to either move the objects in the scene along with the character when they reach the end of the screen (scrolling scene) or render very long stretches of terrain and move the camera along with the character when they reach the end of the screen (scrolling camera).

### 4.1.2.4 Design Constraints

Entire game will be rendered in 2 dimensions, but we must make sure the background is rendered behind all other objects in the scene.

## 4.1.3 Description for Function “drawPlayer”

drawPlayer – render the player in the scene

### 4.1.3.1 Processing narrative (PSPEC) for function n

Function will render the player in the game as they move throughout the scene.

### 4.1.3.2 Function interface description

This function takes no parameters and outputs the character rendering.

### 4.1.3.3 Performance Issues

We must decide at what point we will scroll either the scene or the camera along with the player (ex. when the character is touching the edge of the screen or when the player is approaching the edge of the screen but not touching it yet).

### 4.1.3.4 Design Constraints

Player must be rendered in front of the background and its movement must be constrained by boundaryTests

## 4.1.4 Description for Function “AI”

AI – artificial intelligence for enemy movement and actions

### 4.1.4.1 Processing narrative (PSPEC) for function

Function will dictate how the enemies in the scene act and interact with the player. These enemies will be controlled by algorithms and will not be controlled by players. Their movement will either be a simple back-and-forth movement or they will move towards the player’s position.

### 4.1.4.2 Function interface description

This function takes a time variable as a parameter and changes enemy movement with respect to time.

### 4.1.4.3 Performance Issues

Special performance required for the subsystem is specified.

### 4.1.4.4 Design Constraints

The computer characters will be restricted to movement in certain areas and will move at preset velocities.

## 4.1.5 Description for Function “boundaryTests”

boundaryTests – testing boundaries for movement

### 4.1.5.1 Processing narrative (PSPEC) for function

Function will set up all boundary testing for both character movement and AI enemy movement with objects in the scene and each other.

### 4.1.5.2 Function interface description

Function takes no parameters and simply bounds movement variables like character and AI enemy positions. It does not have any true output.

### 4.1.5.3 Performance Issues

We must keep track of all enemy and character position variables as well as the locations of all objects in the scene to properly test boundaries. This will be done with global variables to reduce the amount of data passed between functions since these functions are called constantly.

### 4.1.5.4 Design Constraints

The boundary numbers will be ballpark numbers which will be tested and perfected to make the smoothest collisions possible.

## 4.1.6 Description for Function “printToScreen”

printToScreen – print bitmap text to the screen

calls drawText to be able to render

### 4.1.6.1 Processing narrative (PSPEC) for function

Function defines what text will be printed to the screen. Most likely some form of directions will be printed at the start of the game, and the player’s score and number of lives will be displayed in the top right corner throughout the duration of the game.

### 4.1.6.2 Function interface description

Function takes no parameters and outputs text to the scene in locations determined by parameters inside this function.

### 4.1.6.3 Performance Issues

We must standardize the screen resolution in which the game will be displayed to ensure the text will appear the same size on every computer.

### 4.1.6.4 Design Constraints

Since we have elected to use bitmap fonts, there are only a select number of font options available in OpenGL/glut. We must be sure to print the text inside the viewing volume of the scene at all times.

## 4.1.7 Description for Function “smoothMoves”

smoothMoves – smooth keyboard-based movement

### 4.1.7.1 Processing narrative (PSPEC) for function

Function will ensure the player’s smooth movement via keyboard input which is slightly different than simply reading if a key has been pressed. We must check to see if the key is being held down or not as well.

### 4.1.7.2 Function interface description

This function takes a time variable as a parameter and changes the character’s position variables smoothly with respect to time and which button on the keyboard is pressed down.

### 4.1.7.3 Performance Issues

Only one button can be detected at a time for smooth movement, which we cannot change.

### 4.1.7.4 Design Constraints

Since only one button at a time can be detected for smooth movement, we will use simple if-else statements to determine what type of variable updating to perform.

## 4.1.8 Description for Function “jump”

jump – make the player jump

would modify variables in drawPlayer

### 4.1.8.1 Processing narrative (PSPEC) for function

Function is called when player presses spacebar. Upon call, player’s vertical position will be moved along a cosine curve (up and back down, simulating gravity) until they land back on a surface (either the ground or a platform).

### 4.1.8.2 Function interface description

The function will take a time variable as input and change the player’s vertical position along the cosine curve with respect to time.

### 4.1.8.3 Performance Issues

We will have to fine tune the mechanics of this function to figure out boundary testing for both landing on platforms and running into the bottom/side of platforms while jumping, which we expect will be quite complicated.

### 4.1.8.4 Design Constraints

Once the function is implemented, we will have to see how left/right movement works while jumping and possibly restrict the amount of left/right movement while in the air.

## 4.1.9 Description for Function “display”

display – render everything in the game

calls all rendering functions as well as AI and boundaryTests

### 4.1.9.1 Processing narrative (PSPEC) for function

Function will both call all rendering functions such as drawScene and drawPlayer as well as call the movement, AI and jump functions. In the OpenGL programming structure, the display method is constantly called. Because of this, it is ideal to make calls to movement functions and other variable-updating functions that must constantly be checked from display.

### 4.1.9.2 Function interface description

The function takes no parameters and outputs nothing directly but constantly calls all rendering and movement functions.

### 4.1.9.3 Performance Issues

Since this function does nothing but call other functions, no performance issues arise.

### 4.1.9.4 Design Constraints

None.

## 4.1.10 Description for Function “idle”

idle – various items to process while game is idle (time variables)

interacts with smoothMoves for time-based movement

### 4.1.10.1 Processing narrative (PSPEC) for function

Function updates all time variables while the game is idle. These variables are called from other functions for uniform movement speed.

### 4.1.10.2 Function interface description

The function takes no parameters and updates global time variables which are used in other functions to generate uniform movement speeds. The smoothMoves function is also called from here, passing a specific frame rate-related time variable to it.

### 4.1.10.3 Performance Issues

The glutGet(GLUT\_TIME\_ELAPSED) function returns units in milliseconds, so to get the time elapsed in seconds we must multiply the input time by (1/1000).

### 4.1.10.4 Design Constraints

To capture the frame rate and allow for uniform movement speeds on all computers running the game, we have set up a delta\_seconds variable which will determine the change in seconds between each frame of the game and make the game run the same speed on all computers.

## 4.1.11 Description for Function “drawText”

drawText – allow text to be rendered and drawn to the screen

### 4.1.11.1 Processing narrative (PSPEC) for function

Function sets up for bitmap text strings to be printed to the screen. printToScreen calls this function with inputs and this function prints them to the screen.

### 4.1.11.2 Function interface description

This function takes position variables and character strings as input and outputs text to the screen at the position that is passed to it.

### 4.1.11.3 Performance Issues

The game must be presented in the same resolution on every screen it is played on to ensure the text appears the same for all players and is not stretched or skewed.

### 4.1.11.4 Design Constraints

As mentioned in printToScreen, only a few different types of bitmap fonts exist so we are limited to the available bitmap fonts supported by OpenGL and glut.

## 4.1.12 Description for Function “special\_down”

special\_down – keep track of when special buttons are held down

special buttons are arrow keys, function keys, etc. (no ASCII value)

### 4.1.12.1 Processing narrative (PSPEC) for function

Function keeps track of when a “special” key (arrow keys, function keys, etc.) is held down on the keyboard.

### 4.1.12.2 Function interface description

This function takes input from the keyboard and stores the key being held down in a global variable.

### 4.1.12.3 Performance Issues

Only “special” keys are detected by this function as specified in glut, so we will need both another function to detect when the special key is released and 2 other functions to detect when regular keys (letters, numbers, etc.) are held down and released.

### 4.1.12.4 Design Constraints

As previously mentioned, only “special” keys can be detected by this function as specified by glut.

## 4.1.13 Description for Function “special\_up”

special\_up – keep track of when special buttons are let go

special buttons are arrow keys, function keys, etc. (no ASCII value)

### 4.1.13.1 Processing narrative (PSPEC) for function

Function keeps track of when a “special” key (arrow keys, function keys, etc.) is let go after being held down on the keyboard.

### 4.1.13.2 Function interface description

This function takes input from the keyboard and stores the key being let go after previously being held down in a global variable.

### 4.1.14.3 Performance Issues

Only “special” keys are detected by this function as specified in glut, so we will need both another function to detect when the special key is held down and 2 other functions to detect when regular keys (letters, numbers, etc.) are held down and released.

### 4.1.14.4 Design Constraints

As previously mentioned, only “special” keys can be detected by this function as specified by glut.

## 4.1.17 Description for Function “keyboardUp”

keyboardUp – keep track of when keyboard buttons are let go

buttons with ASCII values (letters, numbers, etc.)

### 4.1.17.1 Processing narrative (PSPEC) for function

Function keeps track of when a regular key (letters, numbers, etc.) is let go after previously being held down on the keyboard.

### 4.1.17.2 Function interface description

This function takes input from the keyboard and stores the key being let go after previously being held down in a global variable.

### 4.1.17.3 Performance Issues

Only regular keys are detected by this function as specified in glut, so we will need both another function to detect when the regular key is held down and 2 other functions to detect when special keys (arrow keys, function keys, etc.) are held down and released.

### 4.1.17.4 Design Constraints

As previously mentioned, only regular keys can be detected by this function as specified by glut.

## 4.1.18 Description for Function “keyboard”

keyboard – keep track of when keyboard buttons are held down

buttons with ASCII values (letters, numbers, etc.)

changes variables in drawPlayer

### 4.1.18.1 Processing narrative (PSPEC) for function

Function keeps track of when a regular key (letters, numbers, etc.) is held down on the keyboard. Also, certain keys such as the spacebar (“jump” button) and the escape key (which exits the program) are monitored here.

### 4.1.18.2 Function interface description

This function takes input from the keyboard and stores the key being held down in a global variable. If the spacebar is pressed, the jump function is called to make the player jump. If the escape key is pressed, the glut window is destroyed and the program exits properly.

### 4.1.18.3 Performance Issues

Only regular keys are detected by this function as specified in glut, so we will need both another function to detect when the regular key is let go after previously being held down and 2 other functions to detect when special keys (arrow keys, function keys, etc.) are held down and released.

### 4.1.18.4 Design Constraints

As previously mentioned, only regular keys can be detected by this function as specified by glut.

## 4.1.19 Description for Function “CreateGlutWindow”

CreateGlutWindow – create a full screen game window

### 4.1.19.1 Processing narrative (PSPEC) for function

Function performs the necessary glut function calls to create the game window. The mouse cursor is also hidden in this function.

### 4.1.19.2 Function interface description

The function takes no parameters and creates a full screen glut window at a specified resolution depending on the aspect ratio of the user’s screen.

### 4.1.19.3 Performance Issues

The screen resolution must be set here and two of the most common resolutions are the defaults. The function tests to see if the user is on a netbook computer first and sets the resolution accordingly either way.

### 4.1.19.4 Design Constraints

If the screen resolution isn’t an option on the user’s computer, we must test to see if another resolution can be used.

## 4.1.20 Description for Function “CreateGlutCallbacks”

CreateGlutCallbacks – call glut functions constantly for checks

functions called: keyboard, keyboardUp, special\_down, special\_up, idle, display

also allocate memory to store which key is held down

### 4.1.20.1 Processing narrative (PSPEC) for function

Function makes several calls to glut functions so that correct functionality can be distributed properly throughout the program.

### 4.1.20.2 Function interface description

This function takes no parameters and calls several glut functions to run the program correctly, and it also allocates memory to store which key (special and regular) is held down.

### 4.1.20.3 Performance Issues

None.

### 4.1.20.4 Design Constraints

Functions must be called using the correct glut callbacks (such as glutKeyboardFunc(keyboard) to call the correct keyboard function so glut can read from the keyboard).

## 4.1.21 Description for Function “InitOpenGL”

InitOpenGL – clear the background color to white

### 4.1.21.1 Processing narrative (PSPEC) for function

Function clears the background color of the window to white.

### 4.1.21.2 Function interface description

This function takes no parameters and simply outputs a white background in the glut window.

## 4.1.21.3 Performance Issues

None.

### 4.1.21.4 Design Constraints

Additive colors are used to determine which background color will be presented.

## 4.1.22 Description for Function “main”

main – main loop where functions are constantly called for rendering

functions called: CreateGlutWindow, CreateGlutCallbacks, InitOpenGL, initSounds

### 4.1.22.1 Processing narrative (PSPEC) for function

Function calls all the critical functions that make the program work correctly.

### 4.1.22.2 Function interface description

This function loops through constantly to make all the calls the program needs to make to run properly and check for new inputs which then updates variables and renders the new scene.

### 4.1.22.3 Performance Issues

None.

### 4.1.22.4 Design Constraints

The “glutMainLoop()” call must be made to ensure the program constantly loops through this main function (which subsequently calls all other functions) until the user exits the program.

## 4.2 Software Interface Description

The Super Mountaineer Brothers system will have one interface for its users. The system will be viewed through the monitor on the users personal computer.

### 4.2.1 External machine interfaces

There are no external machine interfaces for our system.

### 4.2.2 External system interfaces

There are no external system interfaces for our system.

### 4.2.3 Human interface

The user interface for our system will be displayed in the terminal to the user. The main menu will be a GUI with a button the user will select for the function they wish to perform. The in game interface will be displayed as a 2 dimensional world in which the user inputs commands through the keyboard and the on screen character acts accordingly.

## 4.3 Control flow description

Please reference figures 5 and 6 for models of the control flow of the system.

# 5. Behavioral Model and Description

A description of the behavior of the software is presented.

## 5.1 Description for software behavior

A detailed description of major events and states is presented in this section.

### 5.1.1 Events

A listing of events that will cause behavioral change within the system are presented here.

Menu – This will display a graphical menu to the user, and will display all of the options contained in the system.

New Game – This option will be selected from the menu, and will create a new game session for the user with all values set at default. The default value for extra lives will be 2.

Pause Game – This event will be initiated by pushing the p key on the computer’s keyboard, and will allow the user to pause their current game session. As there will be no data persistence unless the user selects the save game option explicitly, the game will only be paused for as long as the user’s computer is running.

Save Game - Our system will store data as game saves. The game saves will likely be in the form of a simple text file. The game save will contain various player attributes: the brother, lives, level, coins, etc. This option will be selected from the menu, and will save the users game for them.

Resume Game – This event will be initiated either by pushing the ENTER key on the computer’s keyboard (if the game is paused), and by selecting Resume Game from the menu if the user wishes to continue a previously saved game. This will allow the user to resume a previously saved or paused game with game settings as they were at the time the pause occurred.

Quit Game – This option will be selected from the menu, and will allow the user to exit from the game. This will cause all game-related processes running on the computer to terminate.

Sound On/Off – This option will be selected from the menu, and will allow the user to turn the game sounds on or off. The game will not be sound dependant, so this will not affect the playability of the game.

Select Character – This option will allow the user to select either the male Mountaineer character or the female Mountaineer character to use in game-play.

Jump – This event will be initiated by pushing the up arrow on the computer’s keyboard. Jump will cause the playable game character to jump in the 2d space of the game. Jumping is the main way of attacking enemy characters. To kill them, the user must skillfully jump on the heads of the enemy characters.

Move Right – This event will be initiated by pushing and holding the right arrow on the computer’s keyboard. Move Right will cause the playable game character to move to the right in the 2d space of the game for the duration of the key being held down by the user.

Move Left – This event will be initiated by pushing and holding the left arrow on the computer’s keyboard. Move Left will cause the playable game character to move to the left in the 2d space of the game for the duration of the key being held down by the user.

Duck – This event will be initiated by pushing and holding the down arrow on the computer’s keyboard. Duck will cause the playable game character to crouch, or duck, under other objects in game play. The character will remained crouched for as long as the button is pressed by the user.

Go to Menu – This event will be initiated by pushing the Esc key on the computer’s keyboard. Go to Menu will cause the user to be directed to the games main menu, where they may choose what they wish to do from the available choices on the menu.

Throw Item – This event will be initiated by pushing the spacebar on the computer’s keyboard. If the playable game character is in possession of throw-able items, pressing the space bar will throw the items when the space bar is pressed for the duration of the possession of said items.

Touch Enemy Character – Coming into contact with an enemy character other than when jumping on top of them will result in a loss of health and items from the playable character. Depending on the health status of the character, the contact may cause the playable character to die.

Fall Off World – Falling off of the world happens when the playable character falls down a hole in the 2d world. This results in the death of the playable character.

Collect Item – Items will be collected throughout game play. There will be a variety of different items, which will cause different events to happen. Those items are:

* Football Helmet - These items will cause the playable game character to grow in size, increase health, and will allow special items to be collected and utilized.
* Beer Cans – These items will be collected for points as well as extra lives. Collecting 100 of them will grant the playable character an extra life.
* Musket Balls – Collecting this item will enable the playable character to shoot musket balls at enemy characters.
* First Aid Kit – Collecting this item will grant the playable character an extra life.

### 5.1.2 States

A listing of states (modes of behavior) that will result as a consequence of events is presented here.

This section describes the main states of behavior that will result from an action being presented.

Main Menu – Allows the user to choose what they would like to do. The options in the main menu are: New Game, Quit Game, and Sound On/Off.

New Game – The action of the user selecting the new game button from the main menu will result in a new game session being created for the user. All game values will be set to default in this state, and the playable character will be positioned at the start point in the game.

Pause Game – The action of selecting the pause game button from the main menu will cause the application to temporarily stop running, and all aspects of the game will be retained until the user resumes the game. A constraint of this state is that the computer must be kept on for the game data to be retained at the paused position. If the user wishes for the game data to be saved in a file, they must select the Save Game option.

Save Game – The action of saving the game will cause the system to save all vital data to file, and the user will be able to resume their game at a later point in time. The difference between this and Pause Game is that Save Game allows the user to shut off their machine or close the game application, whereas Pause Game will not allow the user to do so.

Resume Game – The action of selecting the resume game button from the main menu will allow the user to resume playing a previously saved game session. Game play will resume precisely where it left off when the game was saved. If the game was paused and not saved, the user will press the ENTER key from the pause screen to continue game-play. Again, the computer must remain on for all time between pausing and resuming the game session.

Game Over – This state occurs when the playable character runs out of lives in the game. The user may choose to start a new game, or exit the game from this state.

Game in Session – This state occurs when the user is currently engaged in a game session. The user may perform all in-game actions in this state, and can press the Esc key to switch to the main menu of the system.

Quit Game - Selecting the Quit Game option from the Menu allows the user to exit from the game system. The user will be prompted with a question to make sure they wish to exit, if they confirm that they wish to exit, then the system will be exited.

Sound On/Off - Selecting the Sound On/Off option from the Menu allows the user to turn the sound on or off in their current game session. The default value for sound will be on.

## 5.2 State Transition Diagrams

State Transitions of the Super Mountaineer Brothers system.

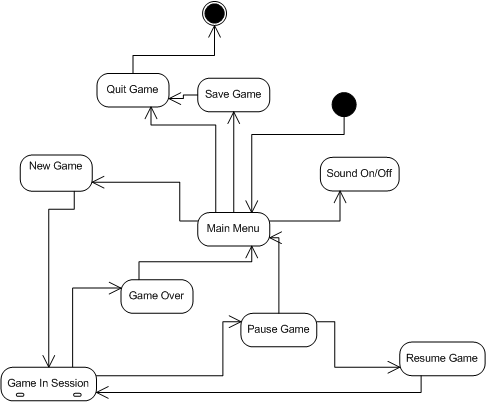


Figure System State Transition Diagram

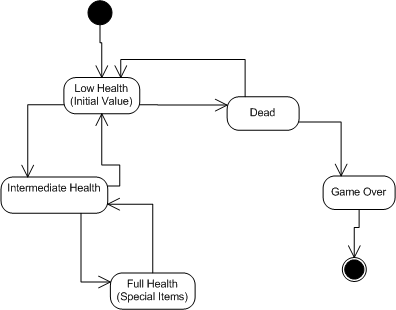


Figure 6 Game in Session State Transition Diagram

## 5.3 Control specification (CSPEC)

The first method where the software manages control is through opening the game application. Once the user is in the game environment, they will be directed to the main menu of the game. They will be presented with the options of “New Game”, “Quit Game”, “Save Game”, and “Sound On/Off”. If the user selects “new Game”, the system will generate a new game session for the user to play. While in game play, the user may press “P” to pause the game. This will cause the system to pause game play. When in the paused state, the user may press “ENTER” to resume game play. At any point during game play or game paused, the user may press “Esc” to be directed to the main menu.

If the user selects “Quit Game” from the main menu, a prompt will be displayed to verify if they wish to quit. If they answer yes, the game application will be exited. If the user selects “Sound On/Off” from the main menu, they will be given the option to turn the game sounds on or off. By default, sound is turned on. If the user selects “Save Game” from the main menu, they will be prompted to make sure they wish to save the current game. If they select yes, the game data will be saved to file, if they select no, the user will be returned to the main menu.

# 6. Restrictions, Limitations, and Constraints

The game application will run on a personal computer, and as such it will be restricted to the limits of that machine. The game will require that the personal computer be running a Windows operating system, XP or later. There must be local storage accessible to the system in order to use the Save Game functionality. Also, the user must have sound drivers installed on the computer in order for the game sound to be played. Also, the user must have a keyboard to play the game.

# 7. Validation Criteria

The validation process is intended to be a way of testing the game application to ensure that all possible errors are caught and handled properly and the system does not crash. The possible error scenarios are listed with their appropriate response in the following section.

## 7.1 Classes of tests and Responses

### 7.1.1 Incorrect keyboard input

If the user presses incorrect keys in an attempt to control the game, the system will ignore these keystrokes and wait for a correct key to be pushed.

## 7.2 Performance bounds

Due to the fact that this application is a 2d graphical game, graphical rendering will be limited by the graphics card contained in the computer the game is being played on. The game sounds depend on sound drivers and some sort of speaker being installed on the local machine as well.

# 8. Appendices

## 8.1 Product Strategies

Due to the nature of this game, a traditional product strategy isn’t really applicable. However, if this game were actually marketable, two key factors would determine the success of this game. The Super Mario Bros. franchise has proven to be highly profitable for Nintendo with a loyal fan base that has grown up along side of the game itself. To attract these same gamers, Super Mountaineer Brothers must implement much of the same functionality as the original game.

Of course, no one will want to buy a game that is identical to the original. In order to appeal to its target market of Mountaineer fans nationwide, Super Mountaineer Brothers must also introduce its own unique features that will set it apart from the original. These unique features must encompass both the well-known aspects of WVU, but also those details that are only known amongst true Mountaineers.