SOFTWARE DESIGN SPECIFICATION

**West Virginia University**

**CS430**

**3/8/2010**

Super Mountaineer Brothers

Nick Fleming, Jason Hooks, Anthony Palma, Adam Trainer

*SOFTWARE DESIGN SPECIFICATION*

Contents

[1.0 Introduction 7](#_Toc255779444)

[1.1 Goals and objectives 7](#_Toc255779445)

[1.2 Statement of scope 7](#_Toc255779446)

[1.4 Major constraints 7](#_Toc255779447)

[2.0 Data design 7](#_Toc255779448)

[2.1 Internal software data structure 7](#_Toc255779449)

[2.2 Global data structure 7](#_Toc255779450)

[2.3 Temporary data structure 8](#_Toc255779451)

[2.4 Database description 8](#_Toc255779452)

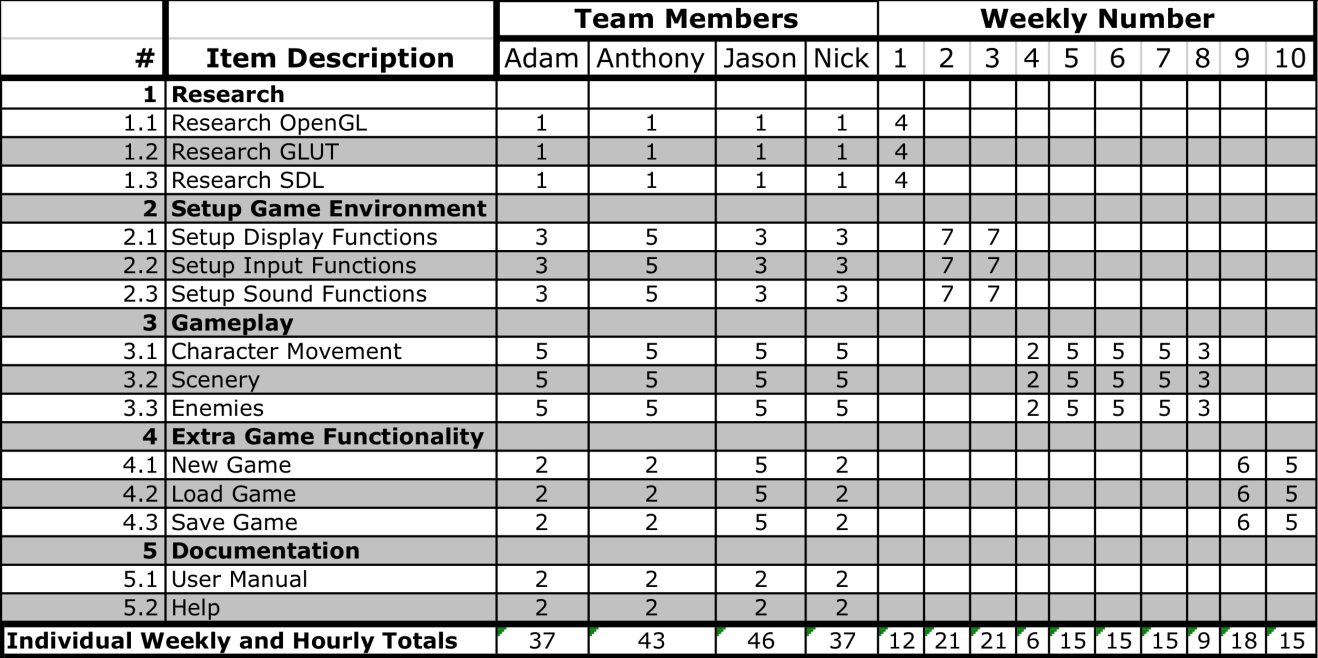
[3.0 Architectural design 8](#_Toc255779453)

[3.1 Program Structure 8](#_Toc255779454)

[3.1.1 Architecture diagram 9](#_Toc255779455)

[4.0 Schedule 10](#_Toc255779456)

[4.1 Scheduling diagram 10](#_Toc255779457)

[ 10](#_Toc255779458)

[4.2 Definition of milestones 10](#_Toc255779459)

[5. Component-Level Design 11](#_Toc255779460)

[5.1 Description for components included in the current design and development iteration of “initSounds” 12](#_Toc255779461)

[5.1.1 Processing narrative (PSPEC) for component 12](#_Toc255779462)

[5.1.2 Component interface description 12](#_Toc255779463)

[5.2 Description for components included in the current design and development iteration of “drawScene” 13](#_Toc255779464)

[5.2.1 Processing narrative (PSPEC) for component 13](#_Toc255779465)

[5.2.2 Component interface description 13](#_Toc255779466)

[5.2.3.5 Design Constraints 13](#_Toc255779467)

[5.3 Description for components included in the current design and development iteration of “drawPlayer” 14](#_Toc255779468)

[5.3.1 Processing narrative (PSPEC) for component 14](#_Toc255779469)

[5.3.2 Component interface description 14](#_Toc255779470)

[5.3.3.5 Design Constraints 14](#_Toc255779471)

[5.4 Description for components included in the current design and development iteration of “AI” 15](#_Toc255779472)

[5.4.1 Processing narrative (PSPEC) for component 15](#_Toc255779473)

[5.4.2 Component interface description 15](#_Toc255779474)

[5.4.3.4 Performance Issues 15](#_Toc255779475)

[5.4.3.5 Design Constraints 15](#_Toc255779476)

[5.5 Description for components included in the current design and development iteration of “boundaryTests” 16](#_Toc255779477)

[5.5.1 Processing narrative (PSPEC) for component 16](#_Toc255779478)

[5.5.2 Component interface description 16](#_Toc255779479)

[5.5.3.4 Performance Issues 16](#_Toc255779480)

[5.5.3.5 Design Constraints 16](#_Toc255779481)

[5.6 Description for components included in the current design and development iteration of “printToScreen” 17](#_Toc255779482)

[5.6.1 Processing narrative (PSPEC) for component 17](#_Toc255779483)

[5.6.2 Component interface description 17](#_Toc255779484)

[5.6.3.5 Design Constraints 17](#_Toc255779485)

[5.7 Description for components included in the current design and development iteration of “smoothMoves” 18](#_Toc255779486)

[5.7.1 Processing narrative (PSPEC) for component 18](#_Toc255779487)

[5.7.2 Component interface description 18](#_Toc255779488)

[5.7.3.4 Performance Issues 18](#_Toc255779489)

[5.7.3.5 Design Constraints 18](#_Toc255779490)

[5.8 Description for components included in the current design and development iteration of “jump” 19](#_Toc255779491)

[5.8.1 Processing narrative (PSPEC) for component 19](#_Toc255779492)

[5.8.2 Component interface description 19](#_Toc255779493)

[5.8.3.4 Performance Issues 19](#_Toc255779494)

[5.8.4.5 Design Constraints 19](#_Toc255779495)

[5.9 Description for components included in the current design and development iteration of “display” 20](#_Toc255779496)

[5.9.1 Processing narrative (PSPEC) for component 20](#_Toc255779497)

[5.9.2 Component interface description 20](#_Toc255779498)

[5.9.3.4 Performance Issues 20](#_Toc255779499)

[5.9.3.5 Design Constraints 20](#_Toc255779500)

[5.10 Description for components included in the current design and development iteration of “idle” 21](#_Toc255779501)

[5.10.1 Processing narrative (PSPEC) for component 21](#_Toc255779502)

[5.10.2 Component interface description 21](#_Toc255779503)

[5.10.3.4 Performance Issues 21](#_Toc255779504)

[5.10.3.5 Design Constraints 21](#_Toc255779505)

[5.11 Description for components included in the current design and development iteration of “drawText” 23](#_Toc255779506)

[5.11.1 Processing narrative (PSPEC) for component 23](#_Toc255779507)

[5.11.2 Component interface description 23](#_Toc255779508)

[5.11.3.4 Performance Issues 23](#_Toc255779509)

[5.11.3.5 Design Constraints 23](#_Toc255779510)

[5.12 Description for components included in the current design and development iteration of “special\_down” 24](#_Toc255779511)

[5.12.1 Processing narrative (PSPEC) for component 24](#_Toc255779512)

[5.12.2 Component interface description 24](#_Toc255779513)

[5.12.3.4 Performance Issues 24](#_Toc255779514)

[5.12.3.5 Design Constraints 24](#_Toc255779515)

[5.13 Description for components included in the current design and development iteration of “special\_up” 25](#_Toc255779516)

[5.13.1 Processing narrative (PSPEC) for component 25](#_Toc255779517)

[5.13.2 Component interface description 25](#_Toc255779518)

[5.13.3.4 Performance Issues 25](#_Toc255779519)

[5.13.3.5 Design Constraints 25](#_Toc255779520)

[5.14 Description for components included in the current design and development iteration of “keyboardUp” 26](#_Toc255779521)

[5.14.1 Processing narrative (PSPEC) for component 26](#_Toc255779522)

[5.14.2 Component interface description 26](#_Toc255779523)

[5.14.3.4 Performance Issues 26](#_Toc255779524)

[5.14.3.5 Design Constraints 26](#_Toc255779525)

[5.15 Description for components included in the current design and development iteration of “keyboard” 27](#_Toc255779526)

[5.15.1 Processing narrative (PSPEC) for component 27](#_Toc255779527)

[5.15.2 Component interface description 27](#_Toc255779528)

[5.15.3.4 Performance Issues 27](#_Toc255779529)

[5.15.3.5 Design Constraints 28](#_Toc255779530)

[5.16 Description for components included in the current design and development iteration of “CreateGlutWindow” 29](#_Toc255779531)

[5.16.1 Processing narrative (PSPEC) for component 29](#_Toc255779532)

[5.16.2 Component interface description 29](#_Toc255779533)

[5.16.3.4 Performance Issues 29](#_Toc255779534)

[5.16.3.4 Design Constraints 29](#_Toc255779535)

[5.17 Description for components included in the current design and development iteration of “CreateGlutCallbacks” 30](#_Toc255779536)

[5.17.1 Processing narrative (PSPEC) for component 30](#_Toc255779537)

[5.17.2 Component interface description 30](#_Toc255779538)

[5.17.3.4 Performance Issues 30](#_Toc255779539)

[5.17.3.5 Design Constraints 30](#_Toc255779540)

[5.18 Description for components included in the current design and development iteration of “InitOpenGL” 31](#_Toc255779541)

[5.18.1 Processing narrative (PSPEC) for component 31](#_Toc255779542)

[5.18.2 Component interface description 31](#_Toc255779543)

[5.18.3.4 Performance Issues 31](#_Toc255779544)

[5.18.3.5 Design Constraints 31](#_Toc255779545)

[5.19 Description for components included in the current design and development iteration of “main” 32](#_Toc255779546)

[5.19.1 Processing narrative (PSPEC) for component 32](#_Toc255779547)

[5.19.2 Component interface description 32](#_Toc255779548)

[5.19.3.4 Performance Issues 32](#_Toc255779549)

[5.19.3.5 Design Constraints 32](#_Toc255779550)

[6.0 User interface design 33](#_Toc255779551)

[6.1 Description of the user interface 33](#_Toc255779552)

[6.1.1 Screen images 33](#_Toc255779553)

[6.1.2 Objects and actions 35](#_Toc255779554)

[6.2 Interface design rules 36](#_Toc255779555)

[7.0 Restrictions, limitations, and constraints 36](#_Toc255779556)

[8.0 Appendices 36](#_Toc255779557)

[8.1 Packaging and installation issues 36](#_Toc255779558)

[8.2 Supplementary information (as required) 36](#_Toc255779559)

[Figure 1 Architecture Diagram 5](#_Toc255763162)

[Figure 2 Current State of SMB 3](#_Toc255763163)3

[Figure 3 The original SMB 3](#_Toc255763164)4

[Figure 4 Futuristic View of SMB 3](#_Toc255763165)4

# 1.0 Introduction

This section provides an overview of the entire design document. This document describes all data, architectural, interface and component-level design for the software.

## 1.1 Goals and objectives

We plan to make a 2D platformer called “Super Mountaineer Brothers.” The game will be a clone of Super Mario Brothers for the NES.  The player will be able to play the game and have their progress saved to a file as they progress through 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 controls 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.

# 2.0 Data design

This section contains a description of all data structures including internal, global, and temporary data structures.

## 2.1 Internal software data structure

There will be a player class that contains the attributes of the player:  health, lives, and the player’s position.  A set will save the high scores.

## 2.2 Global data structure

There are no global data structures in our system.

## 2.3 Temporary data structure

We will use an XML writer to save games and a reader to load games.

## 2.4 Database description

No databases will be used for our system.

# 3.0 Architectural design

## 3.1 Program Structure

Super Mountaineer Bros. will use a function-based architecture, as the nature of the game is conducive to this architecture. All game functionality will be handled internally.

### 3.1.1 Architecture diagram

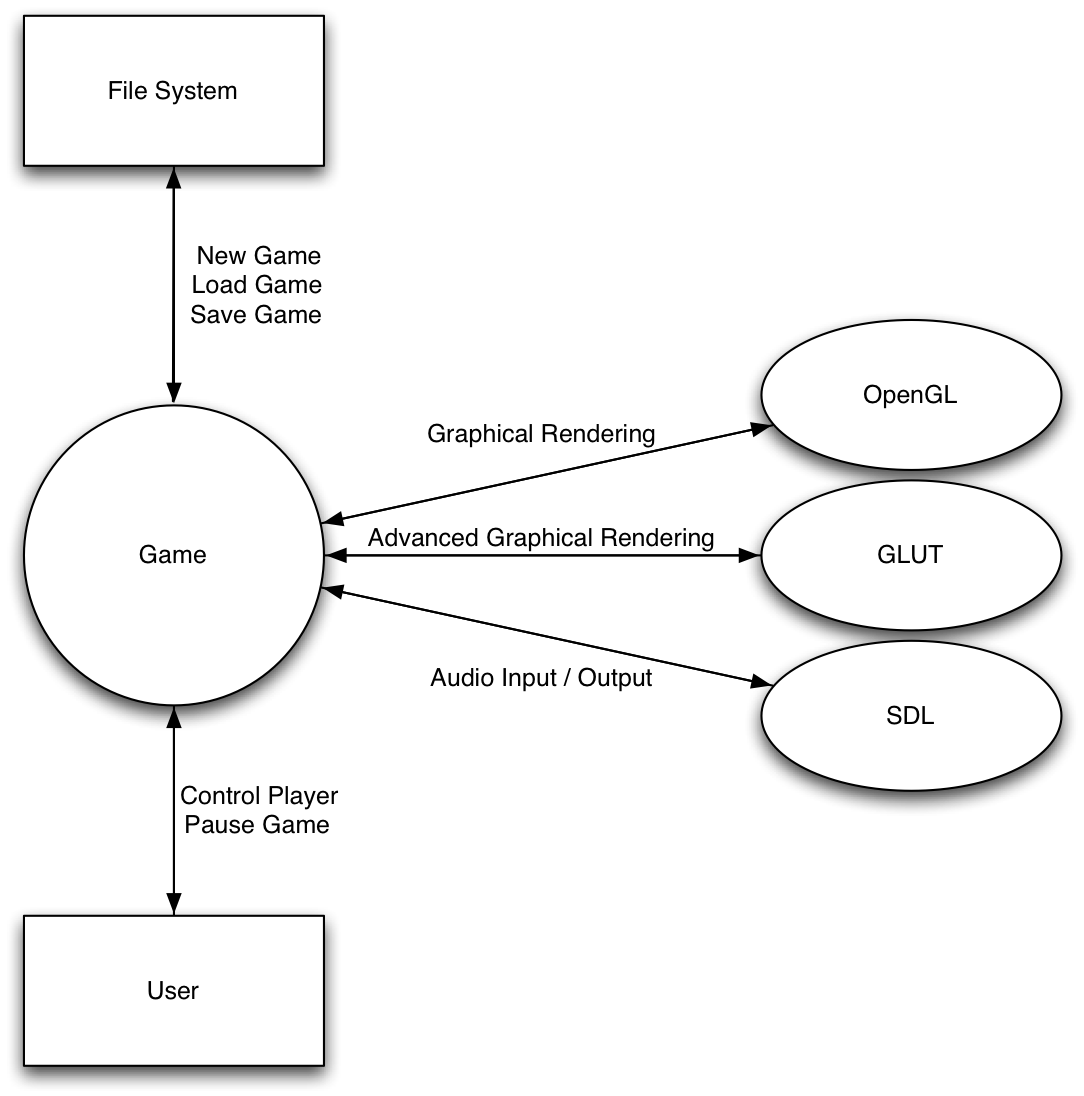


Figure 1 Architecture Diagram

# 4.0 Schedule

## 4.1 Scheduling diagram

## :::::Desktop:Untitled.tiff

Figure Scheduling Diagram

## 4.2 Definition of milestones

|  |  |  |
| --- | --- | --- |
| Milestone Number | Milestone Date | Description of Milestone to be Completed |
| 1 | 3/8/10 | Research |
| 2 | 3/19/10 | Setup Game Environment |
| 3 | 3/26/10 | Character Movement |
| 4 | 3/26/10 | Scenery |
| 5 | 3/26/10 | Enemy AI |
| 6 | 4/8/10 | Extra Game Functionality |
| 7 | 4/8/10 | Finalize Game |
| 8 | 4/15/10 | Documentation |

Table Milestones

# 5. Component-Level Design

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

## 5.1 Description for components included in the current design and development iteration of “initSounds”

initSounds – initialize SDL sound files

### 5.1.1 Processing narrative (PSPEC) for component

Component will initialize the SDL sound components by importing sound files from the source folder

### 5.1.2 Component interface description

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

**5.1.3 Component processing detail**

**5.1.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.1.3.2 Restrictions/limitations**

There are a limited number of chunks available for sound files.

**5.1.3.3 Local data structures**

None.

**5.1.3.4 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.

**5.1.3.5 Design Constraints**

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

## 5.2 Description for components included in the current design and development iteration of “drawScene”

drawScene – draw the scene

### 5.2.1 Processing narrative (PSPEC) for component

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

### 5.2.2 Component interface description

This component takes no parameters and outputs graphical renderings.

**5.2.3 Component processing detail**

**5.2.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.2.3.2 Restrictions/limitations**

None.

**5.2.3.3 Local data structures**

OpenGL drawing structures.

**5.2.3.4 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).

### 5.2.3.5 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.

## 5.3 Description for components included in the current design and development iteration of “drawPlayer”

drawPlayer – render the player in the scene

### 5.3.1 Processing narrative (PSPEC) for component

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

### 5.3.2 Component interface description

This component takes no parameters and outputs the character rendering.

**5.3.3 Component processing detail**

**5.3.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.3.3.2 Restrictions/limitations**

None.

**5.3.3.3 Local data structures**

OpenGL drawing structures.

**5.3.3.4 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).

### 5.3.3.5 Design Constraints

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

## 5.4 Description for components included in the current design and development iteration of “AI”

AI – artificial intelligence for enemy movement and actions

### 5.4.1 Processing narrative (PSPEC) for component

Component 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.

### 5.4.2 Component interface description

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

**5.4.3 Component processing detail**

**5.4.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.4.3.2 Restrictions/limitations**

None (yet).

**5.4.3.3 Local data structures**

None (yet).

### 5.4.3.4 Performance Issues

The AI algorithms will be static paths and will not be randomly generated to simulate the style of Super Mario Bros. where the enemies appear in the same spots every time for familiarity with the level.

### 5.4.3.5 Design Constraints

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

## 5.5 Description for components included in the current design and development iteration of “boundaryTests”

boundaryTests – testing boundaries for movement

### 5.5.1 Processing narrative (PSPEC) for component

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

### 5.5.2 Component interface description

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

**5.5.3 Component processing detail**

**5.5.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.5.3.2 Restrictions/limitations**

None (yet).

**5.5.3.3 Local data structures**

None (yet).

### 5.5.3.4 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 components since these components are called constantly.

### 5.5.3.5 Design Constraints

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

## 5.6 Description for components included in the current design and development iteration of “printToScreen”

printToScreen – print bitmap text to the screen

calls drawText to be able to render

### 5.6.1 Processing narrative (PSPEC) for component

Component 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.

### 5.6.2 Component interface description

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

**5.6.3 Component processing detail**

**5.6.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.1.3.2 Restrictions/limitations**

Only a limited number of bitmap fonts are available.

**5.1.3.3 Local data structures**

Character array.

**5.6.3.4 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.

### 5.6.3.5 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.

## 5.7 Description for components included in the current design and development iteration of “smoothMoves”

smoothMoves – smooth keyboard-based movement

### 5.7.1 Processing narrative (PSPEC) for component

Component 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.

### 5.7.2 Component interface description

This component 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.

**5.7.3 Component processing detail**

**5.7.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.7.3.2 Restrictions/limitations**

The keys must be specified in this function as well as in the keyboard input function.

**5.7.3.3 Local data structures**

None.

### 5.7.3.4 Performance Issues

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

### 5.7.3.5 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.

## 5.8 Description for components included in the current design and development iteration of “jump”

jump – make the player jump

would modify variables in drawPlayer

### 5.8.1 Processing narrative (PSPEC) for component

Component 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).

### 5.8.2 Component interface description

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

**5.8.3 Component processing detail**

**5.8.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.8.3.2 Restrictions/limitations**

None (yet).

**5.8.3.3 Local data structures**

None (yet).

### 5.8.3.4 Performance Issues

We will have to fine tune the mechanics of this component 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.

### 5.8.4.5 Design Constraints

Once the component 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.

## 5.9 Description for components included in the current design and development iteration of “display”

display – render everything in the game

calls all rendering components as well as AI and boundaryTests

### 5.9.1 Processing narrative (PSPEC) for component

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

### 5.9.2 Component interface description

The component takes no parameters and outputs nothing directly but constantly calls all rendering and movement components.

**5.9.3 Component processing detail**

**5.9.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.1.3.2 Restrictions/limitations**

None. Only calls other functions.

**5.1.3.3 Local data structures**

None. Only calls other functions.

### 5.9.3.4 Performance Issues

Since this component does nothing but call other components, no performance issues arise.

### 5.9.3.5 Design Constraints

None.

## 5.10 Description for components included in the current design and development iteration of “idle”

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

interacts with smoothMoves for time-based movement

### 5.10.1 Processing narrative (PSPEC) for component

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

### 5.10.2 Component interface description

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

**5.10.3 Component processing detail**

**5.10.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.10.3.2 Restrictions/limitations**

Time variables are converted from milliseconds to seconds to easily keep track of time.

**5.10.3.3 Local data structures**

None.

### 5.10.3.4 Performance Issues

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

### 5.10.3.5 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.

## 5.11 Description for components included in the current design and development iteration of “drawText”

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

### 5.11.1 Processing narrative (PSPEC) for component

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

### 5.11.2 Component interface description

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

**5.11.3 Component processing detail**

**5.11.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.11.3.2 Restrictions/limitations**

Text, font, and position variables must be passed to the function.

**5.11.3.3 Local data structures**

None.

### 5.11.3.4 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.

### 5.11.3.5 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.

## 5.12 Description for components included in the current design and development iteration of “special\_down”

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

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

### 5.12.1 Processing narrative (PSPEC) for component

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

### 5.12.2 Component interface description

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

**5.12.3 Component processing detail**

**5.12.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.12.3.2 Restrictions/limitations**

None.

**5.12.3.3 Local data structures**

None.

### 5.12.3.4 Performance Issues

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

### 5.12.3.5 Design Constraints

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

## 5.13 Description for components included in the current design and development iteration of “special\_up”

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

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

### 5.13.1 Processing narrative (PSPEC) for component

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

### 5.13.2 Component interface description

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

**5.13.3 Component processing detail**

**5.13.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.13.3.2 Restrictions/limitations**

None.

**5.13.3.3 Local data structures**

None.

### 5.13.3.4 Performance Issues

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

### 5.13.3.5 Design Constraints

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

## 5.14 Description for components included in the current design and development iteration of “keyboardUp”

keyboardUp – keep track of when keyboard buttons are let go

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

### 5.14.1 Processing narrative (PSPEC) for component

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

### 5.14.2 Component interface description

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

**5.14.3 Component processing detail**

**5.14.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.14.3.2 Restrictions/limitations**

None.

**5.14.3.3 Local data structures**

None.

### 5.14.3.4 Performance Issues

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

### 5.14.3.5 Design Constraints

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

## 5.15 Description for components included in the current design and development iteration of “keyboard”

keyboard – keep track of when keyboard buttons are held down

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

changes variables in drawPlayer

### 5.15.1 Processing narrative (PSPEC) for component

Component 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.

### 5.15.2 Component interface description

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

**5.15.3 Component processing detail**

**5.15.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.15.3.2 Restrictions/limitations**

To generate a response from pressing a key on the keyboard, the key must be listed specifically either here or in the smoothMoves function.

**5.15.3.3 Local data structures**

Switch statement.

### 5.15.3.4 Performance Issues

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

### 5.15.3.5 Design Constraints

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

## 5.16 Description for components included in the current design and development iteration of “CreateGlutWindow”

CreateGlutWindow – create a full screen game window

### 5.16.1 Processing narrative (PSPEC) for component

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

### 5.16.2 Component interface description

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

**5.16.3 Component processing detail**

**5.16.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.16.3.2 Restrictions/limitations**

Window size for glutGameMode must be a valid resolution.

**5.16.3.3 Local data structures**

None.

### 5.16.3.4 Performance Issues

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

### 5.16.3.4 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.

## 5.17 Description for components included in the current design and development iteration of “CreateGlutCallbacks”

CreateGlutCallbacks – call glut components constantly for checks

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

also allocate memory to store which key is held down

### 5.17.1 Processing narrative (PSPEC) for component

Component makes several calls to glut components so that correct componentality can be distributed properly throughout the program.

### 5.17.2 Component interface description

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

**5.17.3 Component processing detail**

**5.17.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.17.3.2 Restrictions/limitations**

None. Simply calls other functions.

**5.17.3.3 Local data structures**

None. Simply calls other functions.

### 5.17.3.4 Performance Issues

None.

### 5.17.3.5 Design Constraints

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

## 5.18 Description for components included in the current design and development iteration of “InitOpenGL”

InitOpenGL – clear the background color to white

### 5.18.1 Processing narrative (PSPEC) for component

Component clears the background color of the window to white.

### 5.18.2 Component interface description

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

**5.18.3 Component processing detail**

**5.18.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.18.3.2 Restrictions/limitations**

None.

**5.18.3.3 Local data structures**

None.

## 5.18.3.4 Performance Issues

None.

### 5.18.3.5 Design Constraints

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

## 5.19 Description for components included in the current design and development iteration of “main”

main – main loop where components are constantly called for rendering

components called: CreateGlutWindow, CreateGlutCallbacks, InitOpenGL, initSounds

### 5.19.1 Processing narrative (PSPEC) for component

Component calls all the critical components that make the program work correctly.

### 5.19.2 Component interface description

This component 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.

**5.19.3 Component processing detail**

**5.19.3.1 Algorithmic model (e.g., PDL)**

See Appendix 8.2 for actual code and comments.

**5.19.3.2 Restrictions/limitations**

None. Simply calls other functions.

**5.19.3.3 Local data structures**

None. Simply calls other functions.

### 5.19.3.4 Performance Issues

None.

### 5.19.3.5 Design Constraints

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

# 6.0 User interface design

## 6.1 Description of the user interface

The user will interface with the game through the keyboard and possibly the mouse. The input to the keyboard will be reflected in the graphical glut window on the computer’s screen.

### 6.1.1 Screen images

Figure 1, shown below, is what Super Mountaineer Brothers looks like currently. As you can see, there is still a lot of work to be done. Figure 2 shows an image from the original Super Mario Brothers game, which is our inspiration for this project. Figure 3 shows something similar to what we envision our end product being, although the actual product will not look near as crude as the image in Figure 3.

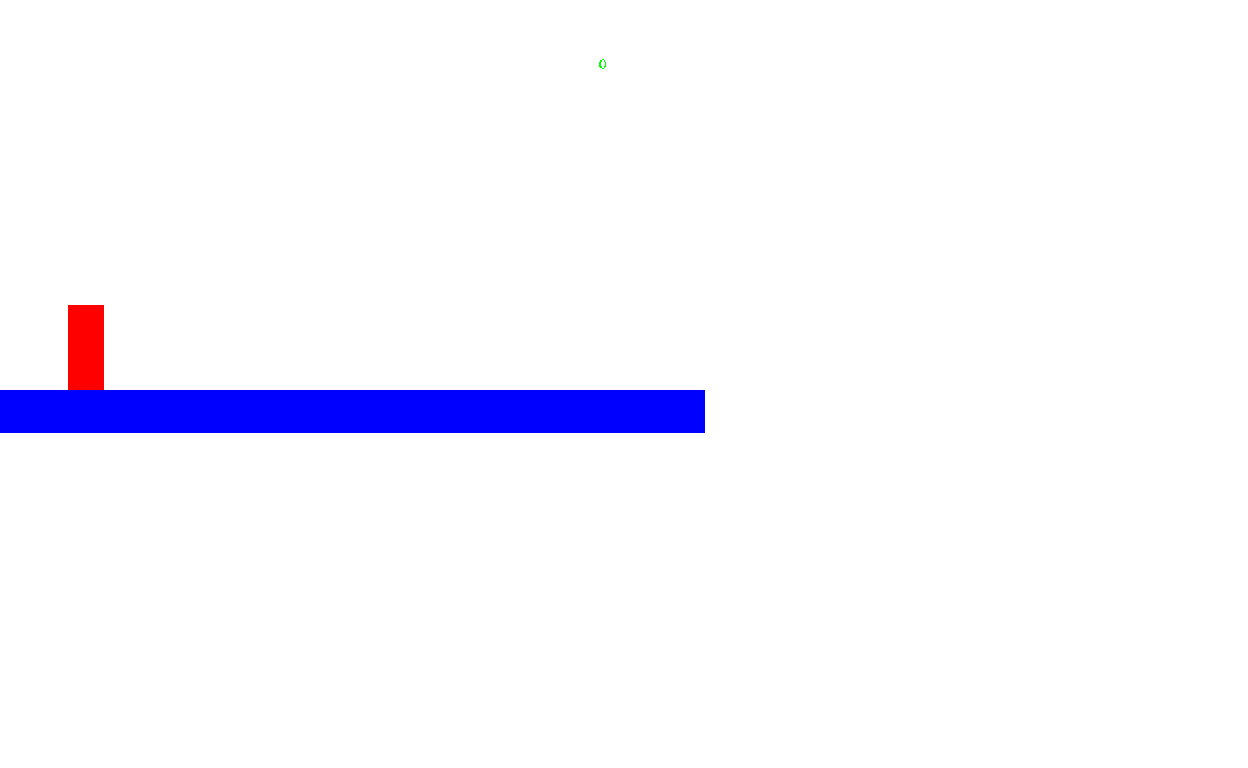
****

Figure 3 Current State of SMB



Figure 4 The original SMB



Figure 5 Futuristic View of SMB

### 6.1.2 Objects and actions

This section identifies all screen objects and actions. The following are the screen objects in our system:

* **Background –** The background of the game will be a static image, it will be applied as an openGL texture.
* **Platforms –** There will be multiple platforms in the game, just as there are in the Super Mario Brothers game. These will be static blocks with a texture wrapping to enhance the appearance. The platforms will allow the user to walk on them, and there will be certain sections that the user may destroy which may house special items. Examples can be seen in Figure 3.
* **Playable character –** The playable character will how the user interacts with the game. The character will respond to events generated by pushing keys on the keyboard.
* **Ground** – The ground will be similar to the platforms in the game, it will also be statically created and have a texture wrapping for enhanced appearance. The ground can be seen in Figure 3, without a texture applied to it.
* **Enemies –** Enemies in the game will be represented by non-playable characters. The purpose of the enemies is to make completion of the level harder for the user. The user must either kill the enemy characters as they are encountered, or must find a way around them.
* **Menu –** The menu is invoked by the user pressing the Escape key on the keyboard. The menu will be displayed in the middle of the screen and contain numerous options to the user, such as ‘Quit Game’, ‘Resume Game’, ‘Sound Options’, and ‘Save Game’.

The following are the screen actions in our system:

* **Run Left –** Pushing the left arrow key on the keyboard will cause the playable character to run left on the screen.
* **Run Right –** Pushing the right arrow key on the keyboard will cause the playable character to run right on the screen.
* **Jump –** Pushing the up arrow on the keyboard will cause the playable character to jump on the screen.
* **Duck –** Pushing the down arrow on the keyboard will cause the playable character to duck on the screen.
* **Throw Object –** Pushing the space bar button on the keyboard will cause the playable character to throw special items, if they are available at the time.
* **Kill Enemy –** Killing an enemy is accomplished by jumping on their heads. The user must exercise caution though, as missing ever so slightly can instead cause them to die.
* **Collect Object –** Collecting objects is achieved by running or jumping into the objects as they are displayed on the screen.
* **Open Menu –** Pushing the Escape key on the keyboard on the keyboard during game-play will open the menu described above.

## 6.2 Interface design rules

The conventions and standards used for designing and implementing the user interface are the following:

We are using Glut to get user input from the keyboard, and possibly the mouse. To display the changes in the system as a result of input from the user, we are using openGL to render a graphical display of the system as the user interacts with it.

# 7.0 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.

# 8.0 Appendices

## 8.1 Packaging and installation issues

Our project will be released to users as a folder containing an exe and libraries.  The user will launch the game through the exe.

## 8.2 Supplementary information (as required)

The first stages of our main code, SMB.cpp, is attached on the following pages. Each function is adequately commented and can be referenced easily to see what actions the function performs.

/\*

CS430 Project

Super Mountaineer Bros.

Nick Fleming

Jason Hooks

Anthony Palma

Adam Trainer

v0.1

\*/

#ifdef WIN32

#include <windows.h>

#endif

#include <stdlib.h>

#include <iostream>

#include <fstream>

//#include <SDL.h>

//#include <SDL\_mixer.h>

#include "SDL\_mixer.h"

#include "SDL.h"

#ifdef \_\_APPLE\_\_

#include <GLUT/glut.h>

#include <OpenGL/gl.h>

#include <OpenGL/glu.h>

#else

#include <GL/glut.h>

#include <GL/glu.h>

#include <GL/gl.h>

#endif

#include <math.h>

using namespace std;

void DrawText(float x, float y, float z, void\* font, char\* text);

void reset(void);

void newAngle(float, int);

static int win = 0;

#define PI 3.1415926535897932384

/\*global variables\*/

int KeyDown[256], SpecialDown[256]; //arrays to store keys for press/release checks

int last\_time = 0; //variable for delta\_time calculations

float proj\_pos[2] = {0.0f, 0.0f}; //projectile position, initially at the origin

float proj\_vel = 1.0f, AI\_vel = 1.1f, player\_vel = 1.0f; //projectile velocity when fired and A.I. paddle velocity

float player[2] = {0.0f, 0.0f}; //player position, initially at the origin

float crosshairs[2] = {0.0f, 0.0f}; //crosshairs position, initially at the origin

int win\_pos[2] = {0, 0}; //window position used for printing window coordinates

float angle = 0.0; //angle of the projectile between the crosshairs and the player

int moving = 0; //used as a boolean variable to determine whether or not the projectile is in motion

int passed\_through = 0; //used as a boolean to determine whether or not the projectile has passed through the crosshairs

float leftPadY = 0.0f, rightPadY = 0.0f;//sets Y coordinates of center of paddles

int p\_score = 0; //player score

bool isVelNeg = 0, server = 0;//booleans to keep track of direction of velocity and serve (0 for right, 1 for left)

bool newGame = 0;//booleans to keep track of welcome and directions messages, and number of players

int channel = -1;

Mix\_Chunk\* hit = NULL;

Mix\_Chunk\* miss = NULL;

/\*initialize the sound effects\*/

void initSounds(void)

{

if (Mix\_OpenAudio(22050, MIX\_DEFAULT\_FORMAT, 2, 4096) != -1)

{

hit = Mix\_LoadWAV("pong2.wav");

miss = Mix\_LoadWAV("pongMiss.wav");

}

}

void drawScene(void)

{

/\*figure out how to move the ground and the background with the player\*/

glColor3f(0.0f, 0.0f, 1.0f);

glBegin(GL\_QUADS);

/\*ground\*/

glVertex3f(-1.0f, -0.8f, 0.0f);

glVertex3f(-1.0f, -1.0f, 0.0f);

glVertex3f(10.0f, -1.0f, 0.0f);

glVertex3f(10.0f, -0.8f, 0.0f);

glEnd();

glColor3f(1.0f, 1.0f, 1.0f);

glBegin(GL\_QUADS);

/\*background (will be textured later)\*/

glVertex3f(-1.0f, -0.8f, 0.0f);

glVertex3f(-1.0f, 1.0f, 0.0f);

glVertex3f(10.0f, 1.0f, 0.0f);

glVertex3f(10.0f, -0.8f, 0.0f);

glEnd();

}

void drawPlayer(void)

{

/\*player is just a quad for now, will change later\*/

glColor3f(1.0f, 0.0f, 0.0f);

glBegin(GL\_QUADS);

glVertex3f(-0.8f + player[0], -0.8f + player[1], 0.0f);

glVertex3f(-0.8f + player[0], -0.4f + player[1], 0.0f);

glVertex3f(-0.7f + player[0], -0.4f + player[1], 0.0f);

glVertex3f(-0.7f + player[0], -0.8f + player[1], 0.0f);

glEnd();

}

/\*Artificial Intelligence code for computer paddle in 1-player games\*/

void AI(float delta\_seconds)

{

//if(moving==1)//if the ball is moving

//{

// if(isVelNeg==0)//if the ball is moving towards the computer paddle (on the right)

// {

// if(rightPadY - proj\_pos[1] > 0.05)

// rightPadY -= AI\_vel \* delta\_seconds;

// else if(rightPadY - proj\_pos[1] < -0.05)

// rightPadY += AI\_vel \* delta\_seconds;

// }

//}

}

/\*boundary testing for various aspects of the game as defined below\*/

void boundaryTests(void)

{

///\*BOUNDARY TESTING (PONG BALL AND COURT)\*/

// if(proj\_pos[0] < -1.0)

// {

// Mix\_PlayChannel(1, miss, 0);

// rightScore++;

// AI\_vel -= 0.075;//this gives the A.I. dynamic difficulty. if you lose a point, the A.I. paddle slows down slightly.

// if(AI\_vel < 0.1)

// AI\_vel = 0.1;

// server = 1; //server to loser, which is left player

// reset();

// }

// else if(proj\_pos[0] > 1.0)

// {

// Mix\_PlayChannel(1, miss, 0);

// leftScore++;

// AI\_vel += 0.075;//this gives the A.I. dynamic difficulty. if you win a point, the A.I. paddle speeds up slightly.

// server = 0; //serve to loser, which is right player

// reset();

// }

// else if(proj\_pos[1] < -0.85 && proj\_pos[0] > -0.87 && proj\_pos[0] < 0.87)

// {

// Mix\_PlayChannel(-1, hit, 0);

// proj\_pos[1] = -0.85;//assures this action is only performed once

// angle = (2\*PI) - angle;

// }

// else if(proj\_pos[1] > 0.85 && proj\_pos[0] > -0.87 && proj\_pos[0] < 0.87)

// {

// Mix\_PlayChannel(-1, hit, 0);

// proj\_pos[1] = 0.85;//assures this action is only performed once

// angle = (2\*PI) - angle;

// }

// /\*BOUNDARY TESTING (PADDLES AND COURT)\*/

// if(leftPadY > 0.77)

// leftPadY = 0.77;

// else if(leftPadY < -0.77)

// leftPadY = -0.77;

// if(rightPadY > 0.77)

// rightPadY = 0.77;

// else if(rightPadY < -0.77)

// rightPadY = -0.77;

// /\*BOUNDARY TESTING (PADDLES AND PONG BALL)\*/

// //left paddle

// if(proj\_pos[0] > -0.92 && proj\_pos[0] < -0.875 && proj\_pos[1] < (+0.135 + leftPadY) && proj\_pos[1] > (-0.135 + leftPadY))

// {

// Mix\_PlayChannel(-1, hit, 0);

// isVelNeg = 0;

// if(angle > PI)

// angle -= PI;

// else

// angle += PI;

// newAngle(leftPadY, 0);

// proj\_pos[0] = -0.87;

// //proj\_vel \*= 1.0325

// proj\_vel += 0.075;

// }

// //right paddle

// else if(proj\_pos[0] > 0.875 && proj\_pos[0] < 0.92 && proj\_pos[1] < (+0.135 + rightPadY) && proj\_pos[1] > (-0.135 + rightPadY))

// {

// Mix\_PlayChannel(-1, hit, 0);

// isVelNeg = 1;

// if(angle > PI)

// angle -= PI;

// else

// angle += PI;

// newAngle(rightPadY, 1);

// proj\_pos[0] = 0.87;

// //proj\_vel \*= 1.0325;

// proj\_vel += 0.075;

// }

}

/\*define what text to be drawn to the screen, including directions and scores\*/

void printToScreen(void)

{

char Score[100];

sprintf(Score, "%d", p\_score);

glColor3f(0.0f, 1.0f, 0.0f);

DrawText(0.7f, 0.7f, 0.0f, GLUT\_BITMAP\_TIMES\_ROMAN\_24 , Score);

}

/\*generate smooth keyboard-based movement\*/

void smoothMoves(float delta\_seconds)

{

/\*if ( KeyDown['d'] || KeyDown['D'] )

player[0] += player\_vel \* delta\_seconds;

if ( KeyDown['a'] || KeyDown['A'] )

player[0] -= player\_vel \* delta\_seconds;

if(onePlayer == 0)

{\*/

if ( SpecialDown[GLUT\_KEY\_RIGHT] )

player[0] += player\_vel \* delta\_seconds;

if( SpecialDown[GLUT\_KEY\_LEFT] )

player[0] -= player\_vel \* delta\_seconds;

//}

}

/\*display function which calls rendering functions for objects in the scene\*/

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

/\*RENDER OBJECTS IN SCENE\*/

drawScene();

drawPlayer();

/\*PERFORM BOUNDARY TESTING\*/

boundaryTests();

/\*PRINT SCORES AND DIRECTIONS TO SCREEN\*/

printToScreen();

glutSwapBuffers();

}

/\*when a point is scored, reset the ball to the origin, reset the ball's velocity, and set the "is ball moving?" variable to false\*/

void reset(void)

{

/\*proj\_pos[0] = 0.0;

proj\_pos[1] = 0.0;

proj\_vel = 1.0;

moving = 0;\*/

}

/\*change the angle when a collision occurs\*/

void newAngle(float padY, int side)

{

float diff;

if(side == 0)//if we're dealing with the left paddle

diff = padY - proj\_pos[1];

else if(side == 1)//if we're dealing with the right paddle

diff = proj\_pos[1] - padY;

if(diff == 0)//avoid a divide-by-zero error

diff = 0.00001;

float x = (0.135/diff) \* 3;

float ang2 = (angle + (PI/x)) \* -1;

/\*keep angle between 0 and 2\*PI for easier calculations\*/

if(ang2 > (2\*PI))

angle = ang2 - (2\*PI);

else if(ang2 < 0)

angle = ang2 + (2\*PI);

else

angle = ang2 \* -1;

/\*limit the angle of the ball to +/- 60 degrees from the horizontal\*/

if(angle > (PI/3) && angle <= PI && isVelNeg == 0)

angle = PI/3;

else if(angle < ((5\*PI)/3) && angle >= PI && isVelNeg == 0)

angle = (5\*PI)/3;

else if(angle < ((2\*PI)/3) && angle >= 0 && isVelNeg == 1)

angle = (2\*PI)/2.75;

else if(angle > ((4\*PI)/3) && angle <= (2\*PI) && isVelNeg == 1)

angle = (4\*PI)/3;

}

/\*update the projectile's position\*/

void idle()

{

int time = glutGet(GLUT\_ELAPSED\_TIME);

int elapsed = time-last\_time;

float delta\_seconds = 0.001f\*elapsed;

last\_time = time;

//proj\_pos[0] += proj\_vel \* moving \* cos(angle)\*delta\_seconds;

//proj\_pos[1] += proj\_vel \* moving \* sin(angle)\*delta\_seconds;

//if(onePlayer == 1)

//AI(delta\_seconds);

smoothMoves(delta\_seconds);

glutPostRedisplay();

}

/\*draw test to the screen\*/

void DrawText(float x, float y, float z, void\* font, char\* text)

{

int len, i;

glRasterPos2f(x, y);

len = (int) strlen(text);

for (i = 0; i < len; i++) {

glutBitmapCharacter(font, text[i]);

}

}

/\*function to utilize arrow keys for a 2-player game\*/

void special\_down(int key, int x, int y)

{

SpecialDown[key] = 1;

}

/\*function to check when an arrow key is released\*/

void special\_up(int key, int x, int y)

{

SpecialDown[key] = 0;

}

/\*create the glut window for display\*/

void CreateGlutWindow()

{

glutInitDisplayMode (GLUT\_DOUBLE | GLUT\_RGBA);

/\*WINDOW MODE (COMMEND OUT FULLSCREEN MODE AND UNCOMMENT THIS TO USE WINDOW MODE)\*/

//glutInitWindowPosition (5, 5);

//glutInitWindowSize (1024, 768); //changed window size to 1024x768

//win = glutCreateWindow ("Ponger");

/\*FULLSCREEN MODE (COMMENT OUT WINDOW MODE AND UNCOMMENT THIS TO USE FULLSCREEN MODE)\*/

glutGameModeString( "1024x600:32@60" ); //the settings for fullscreen mode for netbooks

if (glutGameModeGet(GLUT\_GAME\_MODE\_POSSIBLE))

glutEnterGameMode();

else

{

glutGameModeString("1024x768:32@60");//the settings for fullscreen mode for regular displays

glutEnterGameMode();

}

ShowCursor(FALSE); //hides the mouse cursor

}

/\*function to check when a keyboard button is released\*/

void keyboardUp(unsigned char key, int x, int y)

{

KeyDown[key] = 0;

}

/\*read and interpret keyboard input for various functions\*/

void keyboard(unsigned char key, int x, int y)

{

KeyDown[key] = 1;

// printf("%s(key=%c (ASCII code = %d) , x=%d, y=%d)\n", \_\_FUNCTION\_\_, key, key, x, y);

switch(key){

case 27: //ASCII for Esc key

/\*if the Escape key is pressed, the program will exit\*/

glutLeaveGameMode();

Mix\_CloseAudio();

Mix\_FreeChunk(hit);

exit(0);

break;

case 32: //ASCII for Space Bar

/\*figure out how to make character jump\*/

break;

case 13: //ASCII for Enter

/\*?\*/

break;

}

}

/\*common glut function callbacks\*/

void CreateGlutCallbacks()

{

glutDisplayFunc (display);

glutIdleFunc (idle);

glutSpecialFunc (special\_down);

glutSpecialUpFunc (special\_up);

memset( KeyDown, 0, sizeof( KeyDown ) );

memset( SpecialDown, 0, sizeof( SpecialDown ) );

glutKeyboardFunc (keyboard);

glutKeyboardUpFunc( keyboardUp );

}

/\*initialize background color\*/

void InitOpenGL()

{

glClearColor(1.0f, 1.0f, 1.0f, 0.0f); //sets background to white

}

/\*main function\*/

int main(int argc, char \*\*argv)

{

glutInit(&argc, argv);

CreateGlutWindow();

initSounds();

CreateGlutCallbacks();

InitOpenGL();

glutMainLoop();

return 0;

}