Pillowpants Software

Software Design Specification

Presented By:

Will Kovalick

Matt Moyles

Tony WhiteTable of Contents

1.0 Introduction 3

1.1 Goals and objectives 3

1.2 Statement of scope 3

1.3 Software context 3

1.4 Major constraints 3

2.0 Data design 4

2.1 Internal software data structure 4

2.2 Global data structure 4

2.3 Temporary data structure 4

2.4 Database description 4

3.0 Architectural design 5

3.1 Program Structure 5

4.0 Schedule 6

4.1 Scheduling diagram 6

5.0 Component - level design 7

5.1.1 Description for components included in the current design and development iteration 7

5.1.2 Processing narrative (PSPEC) for state 0 8

5.1.3 Processing narrative (PSPEC) for state 1 10

5.1.4 Processing narrative (PSPEC) for state 2 12

5.1.5 Processing narrative (PSPEC) for state 3 15

5.1.6 Processing narrative (PSPEC) for state 4 17

5.2.1 Processing narrative (PSPEC) for drawFrame0() 18

5.2.2 Processing narrative (PSPEC) for changeSelection(int dir) 19

5.3.1 Processing narrative (PSPEC) for drawFrame1() 20

5.3.2 Processing narrative (PSPEC) for addToParty(int x) 20

5.3.3 Processing narrative (PSPEC) for removeFromParty(int x) 21

5.3.4 Processing narrative (PSPEC) for changeSelection(int dir) 22

5.3.5 Processing narrative (PSPEC) for changeColumn(int dir) 23

5.4.1 Processing narrative (PSPEC) for checkUnitTurn() 23

5.4.2 Processing narrative (PSPEC) for drawFrame2a() 24

5.4.3 Processing narrative (PSPEC) for updateCoord(int dir) 25

5.4.4 Processing narrative (PSPEC) for movePlayer(int x, int y) 26

5.4.5 Processing narrative (PSPEC) for drawFrame2b() 27

5.4.6 Processing narrative (PSPEC) for drawFrame2c 27

5.4.7 Processing narrative (PSPEC) for executeCommand(int com, int x, int y) 28

5.5.1 Processing narrative (PSPEC) for selectTarget() 29

5.5.2 Processing narrative (PSPEC) for findClosePoint() 30

5.5.3 Processing narrative (PSPEC) for chooseCommand() 30

5.6.1 Processing narrative (PSPEC) for drawFrame4() 31

5.6.2 Processing narrative (PSPEC) for exitGame() 32

6.0 User interface design 33

6.1 Description of the user interface 33

6.2 Interface design rules 36

7.0 Restrictions, limitations, and constraints 36

8.0 Appendices 36

8.1 Packaging and installation issues 36

8.2 Software Risk Table 37

## 1.0 Introduction

This product, named “Frat Wars”, will be a tactical RPG (Role Playing Game) that will feature a battle venue and teams comprised of various subjects that are playable by the user. In terms of design of the product, there will need to be data elements to build the game, the ability to handle user input from the keyboard or mouse will be integral in processing the field of battle and to promote an overall satisfactory user experience. This will be accomplished by using an open source game engine and building the game onto this existing infrastructure.

### 1.1 Goals and objectives

The goal of this project is to successfully build a tactical RPG in which a user may control a team of avatars that execute various commands in a turn based battle system. The objective is to complete production of this software in an efficient and error free manner for implementation.

### 1.2 Statement of scope

The product, titled “Frat Wars” will be a tactical RPG that will use existing data within the software to load a battle map based upon collegiate locations within Morgantown. The players will then be loaded on the map and battle will commence. Based upon an artificial intelligence system and user input for the control of the user characters the game will process these commands and subsequently commence the battle until either the user wins or loses.

### 1.3 Software context

The overall context of the overall development of the “Frat Wars” game software is to create an interactive experience for a user. In this case the user is going to be a gaming enthusiast that will want to play the game for an entertainment experience. With the design of Frat Wars we are bringing the user a Tactical RPG game in a collegiate settings.

### 1.4 Major constraints

The only major constraint to consider in the design of this project is within the software being used for the game’s development. We will be using the OpenGL and SDL graphics libraries. Our game is only limited by the rendering constraints of these powerful libraries.

## 2.0 Data design

A description of all data structures including internal, global, and temporary data structures.

### 2.1 Internal software data structure

Data structures that are passed among components the software are described.

Unit – a data structure containing an enumerated value for type(to determine available commands, as well as determining which graphical sprite will be used for display), integer value for health, strength(to determine potency of attacks), and range(to determine how far the character can move during its turn).

Command – a data structure containing an enumerated integer value(to determine which function call is associated with the command. This function call will begin command animation as well as calculate damage or healing results and carryout the change upon the target unit), range(to determine the range of the attack)

### 2.2 Global data structure

Party array – an array of the internal data structure ‘unit’ containing 10 objects of that type. This array is filled during program execution with units chosen by the player. Elements are removed from the array if that object’s ‘health’ attribute is ever less than or equal to 0.

Enemy array – an array of the internal data structure ‘unit’ containing 10 objects of that type. This array is filled during program execution with units chosen at random. Elements are removed from the array if that object’s ‘health’ attribute is ever less than or equal to 0.

### 2.3 Temporary data structure

n/a

### 2.4 Database description

n/a

there will be no traditional database in the sense of a data collection that is updatable or able to be manipulated during execution aside from user command data that is to be accessed on a read-only basis. The specifics of this data is as of now, yet to be determined and is expected to exist only as placeholder data until initial testing is underway. At that time, user command data will be updated and altered as seen fit in order to optimize gameplay and user enjoyment.

Though the number and type of commands is still undecided, one such example may look like the following:

## 3.0 Architectural design

This program will use the Layered Architecture design.

### 3.1 Program Structure

The layers of the architectural design can be broken into 3 levels with each level being dependent on the level directly below itself. The lowest of these levels is the data structure level. This level of the application involves building data structures, at startup, to be filled during program execution. Without these data structures, the next highest level, the processing level, would crash. The data level contains single dimensional arrays containing the selected units for the player team and for the A.I. team. These arrays are filled during program execution.

The processing level will interact back and forth with the preceding level as it fills and manipulates the data structures generated at program startup. Manipulation of these structures is done in the form of adding or removing units from the team arrays as well as manipulating the health values of the unit structures. All other values are read-only. The processing level also handles polling for user input and performing the necessary actions on relevant data, depending on the state of the program.

The final level, the display level, interacts with the lower levels on a strictly read-only basis. At this level, 3D and 2D objects are drawn to the screen. The data structures detailed above each contain data on their type, which is read by the display functions and used to determine what is to be displayed to the screen.

#### 3.1.1 Architecture diagram



Figure 1: architecture diagram

## 4.0 Schedule

### 4.1 Scheduling diagram

**Would have done a Gaant chart here**

* 1. **Definition of milestones**
     1. **Render battleground properly on screen**

The battleground will be properly rendered onto the screen using textured 3D primitives.

*Milestone Complete By: 3/10/2010*

* + 1. **Create types for units and render onto the screen**

The unit class should be finished and a sample unit should be rendered onto the screen. At this point the unit will not be controllable by the user.

*Milestone Complete By: 3/15/2010*

* + 1. **Artwork Design**

Artwork should be finalized at this stage. A different sprite should be created for each type of character. Additionally, several terrain types should be designed which are modeled after terrains you might find common around campus. Examples would include a road, grass, concrete.

*Milestone Complete By: 3/26/2010*

* + 1. **Sample Game Semi-Playable**

At this stage a sample playable game should be hard coded into game. There will be no AI by the computer player hence why it is “Semi-Playable”. This is mainly to test the battle system, movement of characters and path finding from tile to tile.

*Milestone Complete By: 4/2/2010*

* + 1. **Computer Player AI**

A basic computer AI will be developed so the user has an opponent to play against. This AI will be extremely basic and ultimately not very challenging.

*Milestone Complete By: 4/16/2010*

* + 1. **2D Rendering and Menus**

The last milestone will involve all the 2D rendering required for the game. 2D Rendering in our game will encompass the status bar displayed at the bottom of the battleground and the *Start Menu* and *Army Building*  menus displayed to the user before the game starts.

*Milestone Complete By: 4/26/2010*

## 5.0 Component - level design

### 5.1.1 Description for components included in the current design and development iteration

The components of this program are best described in terms of the program states. Each of these states is capable of operating independently until the time comes for transition to another state. Each of these program states operates in the form of a program loop which in most cases begin by polling for user input, calling functions based on that user input, and rendering a new frame. One state, however, does not allow for user input. As such, no polling occurs. In this state, the user input portion of the loop is replaced with the running of A.I. algorithms to determine actions taken by the player’s opponent.

Following is a program flowchart showing transition between all program states as well as major function calls made at each point.

Level 0: program state diagram

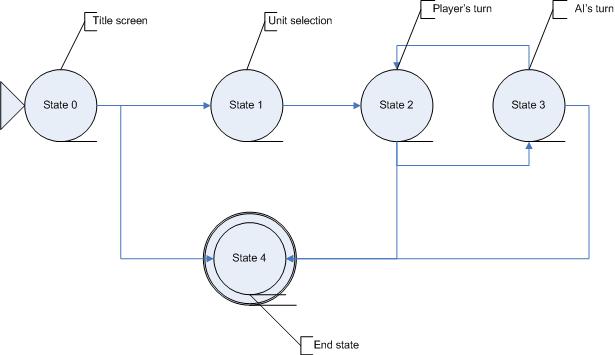


Figure 2: state diagram

### 5.1.2 Processing narrative (PSPEC) for state 0



Figure 3: state 0 flowchart

**5.1.2.1 State 0 interface description.**

Input comes from the keyboard in the form of the up and down directional arrow keys as well as the enter key. Output is both audio and video. Audio is handled in the form of background music that begins playback during module initialization. This initialization occurs where the above flowchart says “BEGIN GAME EXECUTION”. Video output is generated each time the program reaches the point in the flowchart labeled “drawFrame0”

**5.1.2.2 State 0 processing detail**

A detailed algorithmic description for each component is presented. Section 3.2.3 is repeated for each of n components.

**5.1.2.2.1 Interface description**

The user is presented with a simple menu containing 2 options: begin and exit

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

When the user presses an arrow key, the selection variable toggles between 2 values. When the user presses the enter key, the state of the selection variable is used to determine whether program control is transferred to state 1 or state 4.

**5.1.2.2.3 Restrictions/limitations**

Keyboard input other than the up key, down key, and enter key are ignored.

**5.1.2.2.4 Local data structures**

This module is responsible for initializing all data structures to be used in the entire game execution. As none of these data structures are limited to this module, there are no “local” data structures

**5.1.2.2.5 Performance issues**

With extremely little processing to be done in this state, the drawFrame0 function will be called very rapidly causing for a very high framerate.

### 5.1.3 Processing narrative (PSPEC) for state 1



Figure 4: state 1 flowchart

**5.1.3.1 State 1 interface description.**

Initialization of this state does not make changes to the background music initialized in state 0. as such, background music will continue uninterrupted from the previous state. Video output will display to the user consisting of a list of available units for deployment, a list of already selected units, and a text description of the currently highlighted unit

**5.1.3.2 State 1 processing detail**

A loop occurs which is only exited when the user pressed the enter key while the selection variable equals the ‘deploy’ value.

**5.1.3.2.1 Interface description**

The screen is divided into 2 side-by-side halves. The half on the left contains a list of available character units, a counter showing how many characters have been selected in relation to how many can be selected in total, and a text field containing a description of the currently highlighted unit. The right side of the screen contains a list of character units that have been selected by the user as well as a ‘deploy’ button which is activated by navigating to it via the keyboard and pressing the enter key.

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

At the start of the state 1 loop, a poll for user input is made by checking for currently pressed keys. Depending on which, if any, keys are pressed, the corresponding function is called as illustrated in the above flowchart.

**5.1.3.2.3 Restrictions/limitations**

It will be necessary to ensure the user has selected at least 1 unit before state 2 can be initiated. Precautionary measures will need to be put in place to ensure this.

**5.1.3.2.4 Local data structures**

The list of available units will be generated randomly and used by the user to populate the already initialized list of selected units.

**5.1.3.2.5 Performance issues**

No noteworthy performance issues in this module

### 5.1.4 Processing narrative (PSPEC) for state 2

Figure 5: state 2 flowchart

**5.1.4.1 State 2 interface description.**

Input from the user comes in the form of keyboard input. This module accepts input only from the enter key, up key, down key, left key, and right key. Audio output is in the form of sound effects generated during command execution as well as background music initialized during the transition from state 1 to state 2, but not during the transition from state 3 to state 2. This state uses 3 functions for display depending on which point of execution

**5.1.4.2 State 2 processing detail**

This state can be classified as 3 separate states in that execution consists of 3 independent, indefinitely sustainable loops. Program control passes to the next of these loops after the enter key is pressed.

**5.1.4.2.1 Interface description**

The screen displays the battleground and character units at all times. In addition to this, a menu screen will be displayed when the user is expected to select a command. When the user is not expected to enter a command, the user is expected to select a panel on the battleground for one of two purposes. The first of these purposes is to select a position to which the active unit will move. The second of these purposes is to select the panel on which to execute a command.

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

This state begins by determining which player unit is the active unit by deciding which unit is next in a simple queue system. Control then passes to a loop that polls for input and draws a frame. During this loop, the player can make use of the arrow keys to alter the selection coordinates. Pressing the enter key on a valid coordinate (valid coordinate means that the distance from that coordinate to the active unit is less than or equal to the range value of the active unit) will break from the first loop and begin iteration of the second loop. This loop checks only for the up key, down key, and enter key. Up and down cycle through a list of available commands of the active unit. When the user presses enter to confirm the chosen command, control passes to a third loop which behaves similarly to the first loop of this state. The user is expected to select a coordinate on the battleground where the selected command will be executed. If the user’s selected command is “wait”, then this loop will be skipped and program control is passed to state 3.

**5.1.4.2.3 Restrictions/limitations**

n/a

**5.1.4.2.4 Local data structures**

This state must contain variables to determine the user-selected coordinates as well as the user-selected command. These variables are passed as parameters to the movePlayer function as well as the executeCommand function.

**5.1.4.2.5 Performance issues**

This state makes use of 3 distinct functions for drawing a new frame to the screen. As such, each of these functions must be sure to wipe the screen clear before drawing so that image overlap does not occur from the previous drawFrame function.

### 5.1.5 Processing narrative (PSPEC) for state 3

Figure 6: state 3 flowchart

**5.1.5.1 State 3 interface description.**

No user input is accepted during this program state. Output is in the form of existing background music which will have already been initiated earlier during execution as well as sound effects to be played during command execution functions. Video output is in the form of frame drawing functions that exist within the movePlayer and the executeCommand functions.

**5.1.5.2 State 3 processing detail**

Being the only program state to accept no user input, this state is more linear than any other. The computer controlled player follows the same sequence of input that the player uses in state 2. Selection of this input is handled through simple AI functions specified later in this document.

**5.1.5.2.1 Interface description**

The screen displays the battleground, character units, and graphics for command animations. Because there is no user input during this state, no menu or selection icons are to be displayed.

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

Determining which unit is the active unit is performed exactly like it is in state 2, specified above, but with enemy units instead of player units. Target selection is based on a ratio of distance between possible targets to the active unit and remaining health of possible targets. The active unit then moves to the closest possible space to the selected target. Command selection is done by determining which command has the closest maximum range to reach the target. This involves checking all possible commands for the best match. Command execution behaves exactly like it does in state 2, as specified above.

Greater detail on every function of this state is specified below.

**5.1.5.2.3 Restrictions/limitations**

n/a

**5.1.5.2.4 Local data structures**

This state makes use of variables to store the selected coordinates and selected commands to be passed to the movePlayer and executeCommand functions, respectively.

**5.1.5.2.5 Performance issues**

Frame-rate issues are a possibility due to time spent determining the best movement position and the best command choice. This issue will be addressed if testing shows the frame-rate drop to be significant.

### 5.1.6 Processing narrative (PSPEC) for state 4



Figure 7: state 4 flowchart

**5.1.6.1 State 4 interface description.**

The only input accepted in this state is pressing of the enter key. All other keyboard input is ignored. Output is in the form of a single graphic representing success or failure of the battle, or neutral message if this state is accessed directly from state 0.

**5.1.6.2 State 4 processing detail**

This state loops between polling for user input and drawing a frame to the screen until the user presses the enter key. At such time, program execution is terminated.

**5.1.6.2.1 Interface description**

The user is presented with a single graphic showing success or failure of the battle, or a neutral graphic suggesting the player play again in the event that this state is accessed directly from state 0. the user is also presented with a message instructing to press enter to exit the game.

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

n/a

**5.1.6.2.3 Restrictions/limitations**

n/a

**5.1.6.2.4 Local data structures**

n/a

**5.1.6.2.5 Performance issues**

n/a

### 5.2.1 Processing narrative (PSPEC) for drawFrame0()

Sequentially add elements to the screen with no regard for depth buffering, texturing, or shading.

**5.2.1.1 drawFrame0() interface description.**

No input parameters. No return value.

**5.2.1.2 drawFrame0() processing detail**

Sequentially draw the title screen, the menu, and the highlight box to the screen. The position of the highlight box is based on the selection variable.

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

Sequential display objects to the screen.

**5.2.1.2.2 Restrictions/limitations**

Function is dependent on the selection variable containing values 0 or 1

**5.2.1.2.3 Local data structures**

n/a

**5.2.1.2.4 Performance issues**

n/a

### 5.2.2 Processing narrative (PSPEC) for changeSelection(int dir)

This function alters the selection variable by toggling between 0 or 1 depending on the input parameter.

**5.2.2.1 changeSelection(int dir) interface description.**

Input is an integer which is to contain the values 0 or 1. There is no return value

**5.2.2.2 changeSelection(int dir) processing detail**

Selection variable is set equal to the input parameter, if and only if the parameter is 0 or 1.

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

For error checking purposes, ensure the input parameter is 0 or 1. if the value is valid, selection is set equal to the input parameter.

**5.2.2.2.2 Restrictions/limitations**

n/a

**5.2.2.2.3 Local data structures**

n/a

**5.2.2.2.4 Performance issues**

n/a

### 5.3.1 Processing narrative (PSPEC) for drawFrame1()

Draw objects to the screen in sequential order with no regard for depth buffering, texturing, or shading.

**5.3.1.1 drawFrame1() interface description.**

No input parameter. No return value.

**5.3.1.2 drawFrame1() processing detail**

Draw, in sequence, background, menu borders, text description border, unit list text, available unit text, and highlight selection box.

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

Draw objects to the screen in sequence. When drawing unit description, update based on current selection value. When drawing unit list text, update based on unit list data array. When drawing available unit text, update based on available unit data array. When drawing the highlight selection box, update position based on current value of selection variable.

**5.3.1.2.2 Restrictions/limitations**

n/a

**5.3.1.2.3 Local data structures**

n/a

**5.3.1.2.4 Performance issues**

n/a

### 5.3.2 Processing narrative (PSPEC) for addToParty(int x)

The unit data from the available unit data array at position x is copied into the next open slot of the unit list data array, unless the unit list is full.

**5.3.2.1 addToParty(int x) interface description.**

Input is an integer representing the index of the currently highlighted unit from the list of available units. There is no return value

**5.3.2.2 addToParty(int x) processing detail**

If unitlist is not full(as in, if fewer than 10 units exist in the unitlist), the first empty spot is filled with data from availableunits[x]

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

If unitlist is not full(as in, if fewer than 10 units exist in the unitlist), the first empty spot is filled with data from availableunits[x]

**5.3.2.2.2 Restrictions/limitations**

n/a

**5.3.2.2.3 Local data structures**

n/a

**5.3.2.2.4 Performance issues**

n/a

### 5.3.3 Processing narrative (PSPEC) for removeFromParty(int x)

The selected unit is removed from the unitlist array. If this unit is not the last unit in the array, all following units are moved forward in position.

**5.3.3.1 removeFromParty(int x) interface description.**

Input is an integer representing the position in the array to be cleared.

**5.3.3.2 removeFromParty(int x) processing detail**

Unitlist[x] is set to null. If unitlist[x+1] is not null, shift all units forward 1 position.

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

unitlist[x] is set to null. If unitlist[x+1] is not null or beyond the bounds of the array, unitlist[x] is set to unitlist[x+1]. While the next element of unitlist is not null or out of bounds, set each element to the value of the following element.

**5.3.3.2.2 Restrictions/limitations**

n/a

**5.3.3.2.3 Local data structures**

n/a

**5.3.3.2.4 Performance issues**

Removing the element at the head of unitlist[] when unitlist contains its maximum of 10 elements will be the most computationally expensive, but is still expected to execute very quickly.

### 5.3.4 Processing narrative (PSPEC) for changeSelection(int dir)

The selection variable is incremented or decremented depending on the value of dir

**5.3.4.1 changeSelection(int dir) interface description.**

Input is determined by which arrow key the user presses. There is no return value

**5.3.4.2 changeSelection(int dir) processing detail**

If selection is less than 26 and greater than 1, selection is incremented or decremented depending on the value of dir.

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

If selection is less than 26 and greater than 1, selection is incremented or decremented depending on the value of dir. Dir = 0 when the up key is pressed. Dir = 1 when the down key is pressed.

**5.3.4.2.2 Restrictions/limitations**

n/a

**5.3.4.2.3 Local data structures**

n/a

**5.3.4.2.4 Performance issues**

n/a

### 5.3.5 Processing narrative (PSPEC) for changeColumn(int dir)

The selection value is incremented or decremented by 15 depending on the value of dir

**5.3.5.1 changeColumn(int dir) interface description.**

The input parameter determines whether 15 is added to or subtracted from selection

**5.3.5.2 changeColum(int dir) processing detail**

If selection + 15 is less than 26 or selection – 15 is greater than 0, increment or decrement selection by 15 as determined by the value of dir.

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

If selection + 15 is less than 26 or selection – 15 is greater than 0, increment or decrement selection by 15 as determined by the value of dir. Dir = 0 when the left key is pressed. Dir = 1 when the right key is pressed.

**5.3.5.2.2 Restrictions/limitations**

n/a

**5.3.5.2.3 Local data structures**

n/a

**5.3.5.2.4 Performance issues**

n/a

### 5.4.1 Processing narrative (PSPEC) for checkUnitTurn()

Determine which unit is the active unit by checking which unit comes after the previous active unit.

**5.4.1.1 checkUnitTurn() interface description.**

No input parameter. no return value.

**5.4.1.2 checkUnitTurn() processing detail**

Make use of global integer variable activePlayer to determine if unitlist[activeplayer + 1] is available.

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

Make use of global integer variable activePlayer to determine if unitlist[activeplayer + 1] is available. If so, that unit is the new active unit and activePlayer is set to the index of that unit in unitlist[]. If unitlist[activePlayer + 1] is not available(as in, the unit’s health stat is 0 or the end of the array has been reached), the next unit is checked for availability.

**5.4.1.2.2 Restrictions/limitations**

At least one available unit must exist in the unitlist array

**5.4.1.2.3 Local data structures**

n/a

**5.4.1.2.4 Performance issues**

n/a

### 5.4.2 Processing narrative (PSPEC) for drawFrame2a()

Use depth buffering, texture mapping, and shading to render a 3d battleground with a curser hovering above the selected panel.

**5.4.2.1 drawFrame2a() interface description.**

No input parameter. No return value.

**5.4.2.2 drawFrame2a() processing detail**

Use depth buffering, texture mapping, and shading to render a 3d battleground with a curser hovering above the selected panel

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

Use depth buffering, texture mapping, and shading to render a 3d battleground with a curser hovering above the selected panel.

**5.4.2.2.2 Restrictions/limitations**

Function is dependent on values of selectionX and selectionY being within the bounds of the battleground

**5.4.2.2.3 Local data structures**

n/a

**5.4.2.2.4 Performance issues**

n/a

### 5.4.3 Processing narrative (PSPEC) for updateCoord(int dir)

Increment or decrement selectionX or SelectionY depending on the value of dir

**5.4.3.1 updateCoord(int dir) interface description.**

Input parameter corresponds to key pressed by the user. 0 = left, 1 = right, 2 = up, 3 = down.

**5.4.3.2 updateCoord(int dir) processing detail**

Determine which variable to alter, and how to alter by way of a switch case

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

Case dir = 0, selectionX—

Case dir = 1, selectionX++

Case dir = 2, selectionY++

Case dir = 3, selectionY--

**5.4.3.2.2 Restrictions/limitations**

Input parameter must equal 0, 1, 2, or 3

**5.4.3.2.3 Local data structures**

n/a

**5.4.3.2.4 Performance issues**

n/a

### 5.4.4 Processing narrative (PSPEC) for movePlayer(int x, int y)

Build a path from the current location of the active player to the position designated by x and y.

**5.4.4.1 movePlayer(int x, int y) interface description.**

Input parameters represent the x and y coordinates of the desired position on the battleground. There is no return value.

**5.4.4.2 movePlayer(int x, int y) processing detail**

Determine the shortest path to the target, and animate a transition to that point.

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

Using time-based animation, draw the character graphic deltaT/1.5\*distance from its position in the previous frame where deltaT represents the amount of time since the last frame was drawn and distance represents sqrt((x - currentX)^2 + (y – currentY)^2). Using this formula, the character sprite should take 1.5 seconds to move from its current position to the target position.

**5.4.4.2.2 Restrictions/limitations**

The target position, denoted by the parameter coordinates (x, y) should be within activeUnit.range spaces from the active unit.

**5.4.4.2.3 Local data structures**

n/a

**5.4.4.2.4 Performance issues**

n/a

### 5.4.5 Processing narrative (PSPEC) for drawFrame2b()

Use depth buffering, texture mapping, and shading to render a 3d battleground. A 2d command menu is to be displayed on the right-hand side of the screen with a curser pointing to the current selection

**5.4.5.1 drawFrame2b() interface description.**

No input parameter. No return value.

**5.4.5.2 drawFrame2b() processing detail**

Use depth buffering, texture mapping, and shading to render a 3d battleground. A 2d command menu is to be displayed on the right-hand side of the screen with a curser pointing to the current selection

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

Use depth buffering, texture mapping, and shading to render a 3d battleground. A 2d command menu is to be displayed on the right-hand side of the screen with a curser pointing to the current selection

**5.4.5.2.2 Restrictions/limitations**

n/a

**5.4.5.2.3 Local data structures**

n/a

**5.4.5.2.4 Performance issues**

n/a

### 5.4.6 Processing narrative (PSPEC) for drawFrame2c

Use depth buffering, texture mapping, and shading to render a 3d battleground with a curser hovering above the selected panel.

**5.4.6.1 drawFrame2a() interface description.**

No input parameter. No return value.

**5.4.6.2 drawFrame2a() processing detail**

Use depth buffering, texture mapping, and shading to render a 3d battleground with a curser hovering above the selected panel

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

Use depth buffering, texture mapping, and shading to render a 3d battleground with a curser hovering above the selected panel.

**5.4.6.2.2Restrictions/limitations**

Function is dependent on values of selectionX and selectionY being within the bounds of the battleground

**5.4.6.2.3 Local data structures**

n/a

**5.4.6.2.4 Performance issues**

n/a

### 5.4.7 Processing narrative (PSPEC) for executeCommand(int com, int x, int y)

Determine which command function to call and on which position in the battleground.

**5.4.7.1 executeCommand(int com, int x, int y) interface description.**

Com refers to the command. The range of this value is determined by the number of commands are implemented within the program. This number is expected to change throughout program development, but should be approximately 10. x and y refer to the coordinates at which the command should be executed. There is no return value.

**5.4.7.2 executeCommand(int com, int x, int y) processing detail**

Run a switch statement on com to determine which function call corresponds to its value. Pass the x and y parameters directly into that function.

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

Run a switch statement on com to determine which function call corresponds to its value. Pass the x and y parameters directly into that function.

**5.4.7.2.2 Restrictions/limitations**

n/a

**5.4.7.2.3 Local data structures**

n/a

**5.4.7.2.4 Performance issues**

n/a

### 5.5.1 Processing narrative (PSPEC) for selectTarget()

Determine which player target is to be attacked

**5.5.1.1 selectTarget() interface description.**

No input parameters. No return value.

**5.5.1.2 selectTarget() processing detail**

Cycle through all player units to determine the closest 3. of those 3, determine which has the lowest health value.

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

Cycle through all player units to determine the 3 nearest to the active unit. Of those 3, determine which has the lowest health value. That unit is the target unit.

**5.5.1.2.2 Restrictions/limitations**

n/a

**5.5.1.2.3 Local data structures**

nearestUnits[3] – an array of units

**5.5.1.2.4 Performance issues**

n/a

### 5.5.2 Processing narrative (PSPEC) for findClosePoint()

Determine the closest reachable space to the target and call movePlayer(x,y) with that location.

**5.5.2.1 drawFrame2b interface description.**

No input parameters. No return value.

**5.5.2.2 drawFrame2b processing detail**

Using the range attribute of the active unit, count horizontally until lined up with the target’s x position, and then vertically until lined up with the target’s y position until the number of spaces is equal to the range attribute. Call movePlayer(x,y) with the x and y positions of the space that has been reached.

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

Using the range attribute of the active unit, count horizontally until lined up with the target’s x position, and then vertically until lined up with the target’s y position until the number of spaces is equal to the range attribute. Call movePlayer(x,y) with the x and y positions of the space that has been reached.

**5.5.2.2.2 Restrictions/limitations**

n/a

**5.5.2.2.3 Local data structures**

n/a

**5.5.2.2.4 Performance issues**

n/a

### 5.5.3 Processing narrative (PSPEC) for chooseCommand()

All commands have a range inversely proportional to their strength. This function chooses the available command with a range just beyond the distance to the target.

**5.5.3.1 chooseCommand() interface description.**

No input parameters. No return value.

**5.5.3.2 chooseCommand() processing detail**

Count horizontally until lined up with the target, continue counting vertically until the target is met. Cycle through available commands to find the one with the lowest value that is still above that previously counted distance.

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

Calculate distance to the target, determine which command can reach the target with the highest possible damage.

**5.5.3.2.2 Restrictions/limitations**

Target must have been determined by selectTarget() function.

**5.5.3.2.3 Local data structures**

n/a

**5.5.3.2.4 Performance issues**

n/a

### 5.6.1 Processing narrative (PSPEC) for drawFrame4()

Draw conclusion screen and instructions for exiting the game

**5.6.1.1 drawFrame4() interface description.**

No input parameter. no return value

**5.6.1.2 drawFrame4() processing detail**

Draw a ‘win’ or ‘lose’ or ‘forfeit’ image depending on which program state lead to state 4

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

Draw a ‘win’ or ‘lose’ or ‘forfeit’ image depending on which program state lead to state 4

**5.6.1.2.2 Restrictions/limitations**

n/a

**5.6.1.2.3 Local data structures**

n/a

**5.6.1.2.4 Performance issues**

n/a

### 5.6.2 Processing narrative (PSPEC) for exitGame()

Close the program

**5.6.2.1 exitGame() interface description.**

No input parameters. No return value.

**5.6.2.2 exitGame() processing detail**

Call the exit(0) function.

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

n/a

**5.6.2.2.2 Restrictions/limitations**

n/a

**5.6.2.2.3 Local data structures**

n/a

**5.6.2.2.4 Performance issues**

n/a

## 6.0 User interface design

This section of the design document will discuss the user interface for the “Frat Wars” software design and will present various synthesized screenshots of the game while still in development.

### 6.1 Description of the user interface

The user interface will consist of a two dimensional rendering of a game board that will be consistent with what one would view standing at Wallman Hall at West Virginia University and looking across University Avenue to the Mountain Lair. The board will feature rendered tiles in which units may be positioned to attack the AI units. These tiles will be rendered to reflect the sidewalk and scenery that outlays the front of the Mountain Lair. The user will then be given a “pop-up” menu that will display the current unit and any possible actions that may be chosen on that particular turn. (i.e. move, attack, inflict status ailments, heal) The user will be able to select their choice at which time the game will execute this decision and the “pop-up” menu will minimize to show the user the board.

**6.1.1 Screen images**

****

Figure 8: Title screen mockup



Figure 9: Unit selection screen mockup

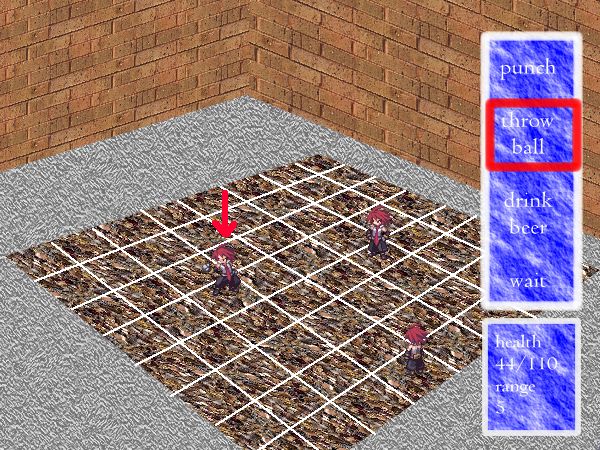


Figure 10: Battle in progress mockup

**6.1.2 Objects and actions**

Main Menu Screen – This screen will feature a window that shows the games title, a logo for the Pillowpants Software Team along with some images promoting the type of game that “Frat Wars” is. There will be a menu on the screen which will feature selection objects for the user. The selections will be listed as “Play” and “Exit”. The user will select one of these choices and based upon the object selected, the game will either advance to a “load out” screen in which players may chose units to use in the game or the game will exit out cleanly and close the window.

Load Out Screen – This screen will be a four paned window that will act as a mechanism for the user to chose which units to select for battle, which units have already been selected, the highlighted units attributes and then a pane that allows the user to deploy their team to load on the following screen for the game board. Objects that will be considered for this screen will be the unit selection, consisting of a data structure for each unit being implemented, an object to display the attributes of the current unit and an event object to load the board and the game play screen once the deploy command has been selected.

Game Play Board Screen – This screen will be a number of rendered objects that comprise the construction of the game board. This will also be the primary window for the game’s functionality. There will be a background that will show the Mountainlair and a number of rendered squares that can be used for player movement and combat. These squares will be internal objects that function as player movement.

### 6.2 Interface design rules

Before the game begins the user will be presented with a menu where they will be prompted to select a set number of units which will make up their army. After they have selected their army, the game will begin and we will move to the “Game in Progress” state.

While the game is in progress we will be rendering the user interface using a tile based system. Each square in the battlefield will be represented by one of these tiles. Only one unit will be able to occupy a tile at any given time.

## 7.0 Restrictions, limitations, and constraints

With the design of this product, there are no operating system or hardware dependent issues to consider when looking at the design. In addition to any of these possibilities, there is no limitation on the breadth of the design based upon the above discussed aspects of the software itself.

In terms of constraints, the time frame in which design, development, testing and implementation must occur for the product to be implemented in a “real world” setting does put a limit on the scope and brevity of the software’s design. Unlike other software development projects, this design must be completed by the specified dead line and there are no instances of time extension for deliverables.

## 8.0 Appendices

### 8.1 Packaging and installation issues

The software itself will be rather small in terms of memory once developed and in terms of packaging the software, the design could be placed on a CD and put in a case for sales, or given the file size of the completed product, there could also be an outlet to download the game and run off of a PC with no installation disk required. The program will run similar to an applet with no in depth installation process to guarantee proper functionality.

In terms of restrictions on installation based upon hardware on the user’s computer, the user will be able to use an Apple, Linux, or Windows platform with no restrictions on video and graphics processing or memory size.

### 8.2 Software Risk Table

**Risk Management Table**

**Frat Wars**

**Date 2010**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Risk #** | **Issue** | **Risk Area** | **Response** | **Impact** | **Probability** |
| R-02 | Procurement of hardware and system software environments | Technical | Project team agrees to use personal computing equipment whenever possible. When not possible, all team members also have access to WVU computing laboratories. | low |  |
| R-04 | Agreement on the software development environment | Technical | Discussion between project team members to agree or compromise on development environment | Low |  |
| R-06 | Requirements change due to time constraints | Technical | Discuss among team members what requirements need to be altered or omitted. | V. High |  |
| **Emerging Risks** | | | | | |
| R-09 | Proprietary constraints on the distribution of source code obtained from third-parties | Technical; Project | Limit use of third-party source code to that with free distribution licensing. | low |  |
| R-11 | Insufficient staffing levels to perform project work | Technical; project | Discuss between project team members ways to alter project scope to fit available workforce | V. High |  |
|  | Software bugs that cannot be solved in time. | Technical; Project | Create workaround that will minimize requirements deviation is all requirements cannot be met. | V. High |  |

**Appendix A**

**Risk Table Explanation Summary**

**Risk Impact Key**

**\*The impact of the risk will never change, but the probability of the risk occurring can change in a**

**given period\***

**Risk Probability Color Key**

The relative likelihood that the risk will occur is low.

The relative likelihood that the risk will occur is high. The risk is imminent.

The relative likelihood that the risk will occur is medium. Proceed cautiously.